# Parametrized Inapproximability Hypothesis under ETH

Venkatesan Guruswami, Bingkai Lin, Xuandi Ren, Yican Sun, Kewen Wu

Presented by Matej Lieskovský

# A glimpse of the destination

## Results

- Proof of Parameterized Inapproximability Hypothesis under Exponential Time Hypothesis - a gap-free assumption.
- A PCP-type theorem for parametrized complexity theory.

We now introduce the necessary terminology.

# Introduction of Problems

## 3SAT

Given a boolean formula in CNF where each clause is limited to at most three literals, decide whether an assignment satisfying all clauses exists.
Example input: $(x \vee y) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (x \vee \neg y \vee z)$

## 2CSP

Given a set of variables, their domains and a set of constraints where each constraint is limited to at most two variables, decide whether an assignment satisfying all constraints exists.
Example input: $x, y \in \{1, \ldots, 100\} \qquad x + y = 4 \quad x - y = 2$

## Gap

A modification of a problem where we are promised that the fraction of satisfiable clauses/constraints is either $1$ or at most $(1 - \varepsilon)$ for some $\varepsilon > 0$

# Introduction - PCP theorem

## Results

- Proof of PIH under ETH - a gap-free assumption.
- A PCP-type theorem for parametrized complexity theory.

## PCP theorem

- Polynomial time reduction from 3SAT to Gap-3SAT
- $NP = PCP[O(\log n), O(1)]$

# Introduction - Parametrized Complexity Theory

## Results

- Proof of PIH under ETH - a gap-free assumption.
- A PCP-type theorem for parametrized complexity theory.

## Parametrized Complexity Theory

- Every instance has a parameter $k$
- We assume $1 \leq k \ll n$
- Time complexity $f(k) \cdot n^{O(1)}$ defines class FPT

## Our target

Consider 2CSP parametrized by the number of variables $k$.
It is known not to lie in FPT (assuming W[1] $\neq$ FPT)

# Introduction - ETH and PIH

## Results

- Proof of PIH under ETH - a gap-free assumption.
- A PCP-type theorem for parametrized complexity theory.

## Exponential Time Hypothesis

Solving 3SAT requires $2^{\Omega(n)}$ time.

## Gap-ETH

Exponential time is also needed for Gap-3SAT.

## Parametrized Inapproximability Hypothesis

No FPT algorithm can solve Gap-2CSP parametrized by number of variables.

# Applications

## $k$-ExactCover

Variant of $k$-SetCover where the $k$ sets selected are disjoint.
Results prove inapproximability under a gap-free assumption.

## Other

- Assuming ETH, no FPT algorithm for $(k, \rho \cdot k)$-ExactCover for any $\rho \geq 1$.
- Assuming ETH, no FPT algorithm for $(k, \rho \cdot k)$-DirectedOddCycleTraversal for some $\rho \in (1, 2)$.

# Discussions

## Previous results

- PIH was only known to hold under Gap-ETH.
- A weaker version was known to hold under W[1] $\neq$ FPT.
- Several potential paths to proving PIH under ETH were proposed but unsuccessful.

## Future directions

- Many parametrized hardness of approximation results can now be based on ETH.
- Further progress toward proving PIH from minimal assumption of W[1] $\neq$ FPT.

# Overview

## Results

- Proof of PIH under ETH - a gap-free assumption.
- A PCP-type theorem for parametrized complexity theory.

We present an efficient reduction from 3SAT formulas to parametrized CSPs of $k$ variables with a constant gap.

## Outline

- Arithmetize 3SAT into an intermediate CSP with $k$ variables and some alphabet size $\Sigma_1$.
- Encode an assignment (proof) using a locally testable and correctable code.
- Check that the input proof is (close to) an encoding of a satisfying assignment.

# Remaining work

## Intermediate parametrized CSP

- Some runtime lower bound under ETH.
- Number of variables is some small $k$.
- Large alphabet size $|\Sigma_1|$

## Error correcting code $C$

- Used to encode a solution of the intermediate CSP.
- Allows a local check of satisfiability.
- Alphabet size polynomial in $|\Sigma_1|$.
- Codeword length arbitrary in $k$ but independent of $|\Sigma_1|$.

# Vectorization

We force $\Sigma_1$ to be a vector space $\mathbb{F}^d$ where $\mathbb{F}$ is a field of constant size. Thus we can view an assignment $\sigma \in \Sigma_1^k = (\mathbb{F}^d)^k$ as $(\mathbb{F}^k)^d$. We will use a code $C : \mathbb{F}^k \to \mathbb{F}^{k'}$ to make coding independent of the entire $\Sigma_1 = \mathbb{F}^d$.
This motivates the design of the intermediate CSP problem. We need:

- ETH-hardness.
- Alphabet is a vector space.
- Restricted constraints for efficient testing.

We call the result Vector-Valued CSPs, which have the following properties:

- We show a reduction from 3SAT.
- $\mathbb{F}$ has characteristic two.
- There are only two constraint types.

# The bulk of the paper

## The theorem

Assume ETH is true. No algorithm can decide $\frac{1}{9600}$-Gap $k$-2CSP within runtime $f(k) \cdot n^{o(\sqrt{\log \log k})}$ for any computable function $f$.

## Structure of the proof

- A detailed definition of Vector-Valued CSPs.
- A reduction from 3SAT to Vector-Valued CSPs.
- A reduction from Vector-Valued CSPs to Gap CSPs.

## The last three sections

- The first reduction
- The second reduction
- Parallel PCPPs

## Thank you for your attention!

A closer look at the use of Vector-Valued CSPs follows, time permitting.

# Formal definition of CSP

We focus on 2CSP exclusively.

A CSP instance $G = (V, E, \Sigma, \{\Pi_e\}_{e \in E}$ where

- $V$ is the set of variables.
- $E$ is the set of constraints. Each constraint $e = \{u_e, v_e\} \in E$ has arity 2 and is related to two distinct variables $u_e, v_e \in V$. The *constraint graph* is the undirected graph on vertices $V$ and edges $E$. Note that we do allow parallel edges.
- $\Sigma$ is the alphabet of each variable in $V$.
- $\{\Pi_e\}_{e \in E}$ is the set of constraint validity functions

# Formal definition of Vector-Valued CSP

A CSP instance $G = (V, E, \Sigma, \{\Pi_e\}_{e \in E})$ is a VecCSP instance if the following additional properties hold:

- $\Sigma = \mathbb{F}^d$ is a $d$-dimensional vector space over a finite field $\mathbb{F}$ with characteristic 2.
- For each $e = \{u, v\} \in E$ where $u = (u_1, \ldots, u_d)$ and $v = (v_1, \ldots, v_d)$ the constraint validity function $\Pi_e$ is either *Linear* or *Parallel*.
- Each variable is related to at most one parallel constraint.

# Formal definition of the constraints

## Linear constraint

There exists a matrix $M_e \in \mathbb{F}^{d \times d}$ such that

$$\Pi_e(u, v) = 1 \iff u = M_e v$$

In our application, $M_e$ will always be a permutation matrix.

## Parallel constraint

There exists a sub-constraint $\Pi_e^{sub} : \mathbb{F} \times \mathbb{F} \to \{0, 1\}$ and a subset of coordinates $Q_e \subset [d]$ such that $\Pi_e(u, v)$ checks $\Pi_e^{sub}$ for every coordinate in $Q_e$, i.e.,

$$\Pi_e(u, v) = \bigwedge_{i \in Q_e} \Pi_e^{sub}(u_i, v_i)$$

# From 3SAT to VecCSP

## Step 1

Divide the clauses and variables into $k$ parts each. Build a "vertex" for each part. Add consistency checks.

## Step 2

Duplicate each vertex into several copies such that

- each vertex is related to at most one constraint
- the sub-constraints inside each constraint form a matching

## Step 3

Rearrange the coordinates of each vertex to make the constraint Parallel. Add a cycle of Linear constraints on the copies of each vertex to ensure consistency.