TENSOR ISOMORPHISMS: NOTES FROM THE SHONAN MEETING - DAY 1

MAKSYM ZUBKOV

Contents

Date: June 4, 2026.

1. 1st Speaker: Tensor Isomorphism – Algorithms and Complexity

Youming Qiao, University of Technology Sydney

Youming Qiao is a professor at the Centre for Quantum Software and Information at the University of Technology Sydney. His work is in theoretical computer science, especially computational complexity theory and algebraic computation, with further connections to cryptography, quantum information and computation, and group theory. This background was very visible in the tutorial: tensor isomorphism was presented not just as one isolated problem, but as a common language for many isomorphism problems appearing in algebra, graph theory, cryptography, quantum information, and computational complexity.

The general theme of the tutorial was that tensor isomorphism provides a unifying framework for several central isomorphism problems. We come from different backgrounds and have different experiences, but a natural journey into this meeting starts with tensor isomorphism.

Let us define the problem.

Definition 1.1. Given $A, B \in U \otimes V \otimes W$, we say that A and B are isomorphic if they lie in the same orbit of

$$GL(U) \times GL(V) \times GL(W)$$

acting naturally on $U \otimes V \otimes W$.

In other words, tensor isomorphism is an orbit problem.

- We are interested in orbit problems for group actions.
- We need efficient representations of both the group G and the set or space S on which it acts.
- Here $GL(U)$ denotes the group of invertible linear transformations of U.

Algorithmically, the problem is:

Input: $A, B \in U \otimes V \otimes W$.
Output: Decide whether there exists

$$(P, Q, R) \in GL(U) \times GL(V) \times GL(W)$$

such that A maps to B.

1.1. **Tensor isomorphism via matrix spaces.** Tensor isomorphism can also be described using matrix spaces.

Given a tensor T_A, take its front slices:

$$F_A = (A_1, \dots, A_n).$$

The linear span of the slices gives a matrix space

$$\mathcal{F}_A = \operatorname{span}(A_1, \dots, A_n) \leq M_{\ell \times m}(\mathbb{F}).$$

Definition 1.2. A tensor A is W-concise if

$$\dim(\mathcal{F}_A) = \dim(W) = n.$$

Lemma 1.3. *If A and B are W-concise, then*

$$A \cong B$$

if and only if the matrix spaces $\mathcal{F}_A$ and $\mathcal{F}_B$ are equivalent, i.e. if and only if there exist invertible matrices L and R such that

$$\mathcal{F}_A = L\mathcal{F}_B R^t = \{LSR^t \mid S \in \mathcal{F}_B\}.$$

In this matrix-space formulation, the $GL(W)$ action is implicit: changing the basis of W changes the list of slices, but not their linear span.

Matrix space equivalence is therefore another formulation of tensor isomorphism. In coding theory, matrix spaces are often called matrix codes.

A useful isomorphism invariant is the rank profile:

$$\{\operatorname{rank}(S) \mid S \in \mathcal{F}_A\}.$$

Over $\mathbb{F}_q$, this can be computed by checking all elements of the matrix space, with running time roughly

$$q^n \operatorname{poly}(n, m, \ell, \log q).$$

The minimum rank

$$\min\{\operatorname{rank}(S) \mid S \in \mathcal{F}_A,\ S \neq 0\}$$

is also an important invariant.

1.2. **Tensor isomorphism, unitary tensor isomorphism, and graph isomorphism.** One recurring theme was the relationship between tensor isomorphism, unitary tensor isomorphism, and graph isomorphism.

1.3. **Tensor isomorphism from GL to U.** Instead of using general linear groups, one can also study tensor isomorphism under groups of unitary matrices.

This setup comes from quantum information theory, where it is known as local unitary equivalence. Pure quantum states can be represented by tensors, and unitary transformations represent legitimate physical operations. Thus, classifying quantum states naturally leads to tensor isomorphism problems under unitary group actions.

1.4. **k-tensor isomorphism reduces to 3-tensor isomorphism.** Given a tensor space

$$V_1 \otimes \cdots \otimes V_k$$

with the natural action of

$$GL(V_1) \times \cdots \times GL(V_k),$$

one can reduce the isomorphism problem for higher-order tensors to the case of 3-tensors.

Theorem 1.4 (Grochow–Qiao)**.** *Given two k-tensors, one can construct in polynomial time two 3-tensors*

$$A, B \in U \otimes V \otimes W$$

such that

- *the original tensors are isomorphic if and only if $A \cong B$;*
- *the total dimension satisfies*

$$\dim(U) + \dim(V) + \dim(W) = O(k^2 \cdot m^{k/2}).$$

Remark 1.5. It is enough, in a precise complexity-theoretic sense, to understand 3-tensors. Understanding tensors of higher order can be reduced to understanding 3-tensors.

Question: Think more about how this reduction works and whether there are useful geometric interpretations of this construction.

1.5. **Graph isomorphism.** There is also a classical construction going from graphs to tensors, going back to ideas of Tutte, Edmonds, and Lovasz.

An edge can be mapped to an elementary matrix. For a bipartite graph

$$G = (L \cup R, E), \qquad |L| = |R| = n, \quad |E| = m,$$

one obtains an $n \times n \times m$ tensor T_G, sometimes called a graphical tensor.

Graph isomorphism can therefore be formulated as a tensor isomorphism problem, especially when the group action is restricted to permutation matrices.

Question: How exactly is the graph tensor constructed from a given graph?

Proposition 1.6 (Li–Qiao–Wigderson–Zhang)**.** *Graphs G and H are isomorphic if and only if their associated graphical tensors T_G and T_H are isomorphic as tensors.*

1.6. **Varying the group types.** One can vary the underlying field and the acting group. For example, over

$$\mathbb{F} = \mathbb{R}, \mathbb{C},$$

one can study actions of

$$U(n), \quad O(n), \quad \mathrm{Sym}(n), \quad Sp(n).$$

Here $\mathrm{Sym}(n)$ can be viewed as the group of permutation matrices.

1.7. **An intricate relationship.** One striking point is that tensor isomorphism and graph isomorphism are closely related, but the relationship is subtle.

Theorem 1.7. *Tensor isomorphism over finite fields reduces to graph isomorphism for graphs of order*

$$q^{O(\ell+m+n)}.$$

Thus tensor isomorphism can be connected back to graph isomorphism, but the reduction can cause a large blow-up in size.

1.8. **Finite group isomorphism.** Another major application is finite group isomorphism: given two finite groups G and H, decide whether they are isomorphic.

A classical algorithm solves this problem in time

$$N^{O(\log N)}$$

for groups of order N.

One important question is how the groups are given in the input:

- Cayley tables;
- permutation groups;

- matrix groups.

The goal is to obtain algorithms that are polynomial in the group order N, or at least substantially faster than the brute-force approach.

The case of p-groups seems especially hard. We will hear more about this later from Bettina Eick.

A first abundant family is given by p-groups of class 2 and exponent p.

For such groups, one can study the commutator map

$$\phi_G : G/[G,G] \times G/[G,G] \to [G,G].$$

Lemma 1.8 (Baer)**.** *For suitable p-groups of class 2 and exponent p, the group isomorphism problem can be translated into the isomorphism problem for the associated commutator map.*

This helps justify why one studies tensors of the form

$$U \otimes V \otimes W$$

rather than only alternating tensors such as

$$U \wedge U \otimes W.$$

1.9. **Polynomial isomorphism.** Suppose that

$$\mathrm{char}(\mathbb{F}) \neq 2, 3.$$

Let f be a cubic form. Associated to f is a symmetric trilinear form

$$T_f : \mathbb{F}^n \times \mathbb{F}^n \times \mathbb{F}^n \to \mathbb{F},$$

obtained by polarization.

The key claim is:

$$f \cong g \quad \Longleftrightarrow \quad T_f \cong T_g.$$

This connection has a cryptographic origin. Patarin proposed using polynomial isomorphism as the security basis for a digital signature protocol.

Three cases are especially studied in multivariate cryptography:

- polynomial isomorphism;
- isomorphism of cubic forms;
- equivalence of multilinear or trilinear forms.

There are two major families of algorithmic techniques:

- algebraic algorithms, such as solving polynomial systems using Gröbner bases;
- graph-theoretic algorithms.

1.10. **A zero-knowledge protocol for graph isomorphism.** A classical example is the zero-knowledge interactive protocol for graph isomorphism.

Consider two players, Alice and Bob. They are given two graphs G and H.

If G and H are isomorphic, honest Alice knows an isomorphism between them.

Alice's goal is to demonstrate that she knows the isomorphism without revealing it to Bob.

Bob's goals are:

- completeness: an honest Alice should convince Bob;
- soundness: a dishonest Alice should not be able to convince Bob unless she really knows an isomorphism.

Question: How does he visualize the hand-drawing example for graph isomorphism?

Theorem 1.9 (Ivanyos–Qiao)**.** *Over $\mathbb{F}_q$, with $\mathrm{char}(\mathbb{F}_q) \neq 2$, the orbit problem for $GL(U)$ acting on*

$$U \odot U \otimes W$$

is in randomized polynomial time.

Theorem 1.10 (Futorny–Grochow–Sergeichuk)**.** *Several orbit problems arising from tensors, symmetric tensors, and related algebraic structures are polynomial-time equivalent. In particular:*

(1) the relevant orbit problems are polynomial-time equivalent;

(2) graph isomorphism reduces to tensor isomorphism;

(3) k-tensor isomorphism reduces to 3-tensor isomorphism.

1.11. **From matrix spaces to matrix tuples.** Given two matrix tuples

$$A = (A_1, \ldots, A_m), \qquad B = (B_1, \ldots, B_m),$$

one can ask whether there exists a change of basis making them simultaneously equivalent, congruent, or conjugate, depending on the group action.

Recall that tensor isomorphism can be formulated through matrix spaces. Thus, matrix tuple problems provide another algorithmic viewpoint on tensor isomorphism.

Question: Show this connection to shallow polynomial neural networks in our work over finite fields.

Xiaorui Sun's algorithm gives a beautiful and intricate reduction from tensor isomorphism to matrix tuple congruence.

A key subroutine is solving tuple congruence.

Module isomorphism is a classical problem in computer algebra. One important tool in this direction is the MeatAxe algorithm.

Matrix tuple conjugacy is linearizable. For example, to decide whether

$$LA_iL^{-1} = B_i$$

for all i, one can write this as

$$LA_i = B_iL$$

for all i.

Similarly, matrix tuple equivalence is also linearizable.

However, matrix tuple congruence is more subtle. This is where methods involving James Wilson and Peter Brooksbank become relevant.

For

$$A = (A_1, \ldots, A_m),$$

the endomorphism algebra is

$$\mathrm{End}(A) = \{E \in M(n, \mathbb{F}) \mid EA_i = A_iE \text{ for all } i \in [m]\}.$$

Other related algebraic structures include:

- endomorphism algebras;
- adjoint algebras;
- centroid algebras;
- Jacobson radical decompositions;
- semisimple algebra decompositions.

1.12. **Orbit closures.** Matrix tuple conjugacy asks: given A and B, does there exist L such that

$$LA_iL^{-1} = B_i$$

for all i?

This problem is linearizable, since it becomes

$$LA_i = B_iL.$$

This leads to the following algorithmic strategy:

- solve the corresponding linear system;
- obtain a basis of the solution space;
- decide whether the solution space contains a full-rank matrix.

More generally, given matrices

$$C_1, \dots, C_\ell,$$

one may ask whether

$$\operatorname{span}(C_1, \dots, C_\ell)$$

contains a full-rank matrix.

This is related to symbolic determinant identity testing.

If $|\mathbb{F}| = O(1)$, this problem is NP-hard. If

$$|\mathbb{F}| \geq 2n,$$

then randomized polynomial-time algorithms follow from the Schwartz–Zippel lemma.

In the 1960s, Edmonds asked for a deterministic polynomial-time algorithm.

A theorem of Kabanets and Impagliazzo says that a deterministic polynomial-time algorithm for symbolic determinant identity testing would imply major algebraic complexity lower bounds.

Now we move from orbits to orbit closures.

The orbit closure intersection problem asks: given a group G acting on a topological or algebraic space S, do the orbit closures of two points $s, t \in S$ intersect?

This is a geometric version of the orbit problem.

For example, for the conjugation action

$$A \mapsto LAL^{-1},$$

orbit classification corresponds to Jordan normal form, while orbit closure information is closer to eigenvalue data.

This is analogous to the philosophy that border rank may be easier to understand than tensor rank.

For a group action, we may ask:

- What are the polynomial invariants?
- What degree bound is needed to generate the invariant ring?
- Can the invariant ring be described succinctly?
- Can the invariants be computed efficiently?

For matrix tuple conjugacy under

$$GL_n$$

acting on

$$(A_1, \dots, A_m),$$

the invariants are trace monomials

$$\operatorname{Tr}(A_{i_1} \cdots A_{i_k})$$

by work of Procesi.

The degree bound is

$$k \leq n^2$$

by Procesi and Razmyslov.

Trace monomials can also be encoded by small algebraic circuits, as in work of Mulmuley.

For tensor isomorphism, for example under

$$SL_n \times SL_n \times SL_n$$

acting on

$$\mathbb{C}^n \otimes \mathbb{C}^n \otimes \mathbb{C}^n,$$

we do not yet have an equally simple and efficient family of polynomial invariants.

The degree bounds are likely to be much larger, possibly exponential, as suggested by work of Derksen.

1.13. **Main takeaway.** The main message of the tutorial is that tensor isomorphism is a central organizing problem. It connects:

- graph isomorphism;
- group isomorphism;
- polynomial and cubic form isomorphism;
- matrix space and matrix tuple equivalence;
- quantum information;
- cryptography;
- invariant theory;
- orbit closures and algebraic geometry.

2. Speaker 2: Primes via Zeros

Nitin Saxena, IIT Kanpur

Primes via zeros: interactive proofs for testing primality of natural classes of ideals

This talk was based on joint work with Abhibhav Garg and Rafael Oliveira, presented at STOC 2025. Nitin Saxena's background is in algebraic and computational complexity theory, with important work on primality testing, polynomial identity testing, algebraic independence, polynomial factoring, and related algebraic problems. This makes the topic of the talk very natural: it studies primality not for integers, but for ideals generated by polynomial equations.

The main question is whether the ideal generated by a given system of polynomial equations is a prime ideal.

This problem is too hard in full generality, so the goal is not to solve it for all ideals. Instead, the talk focuses on certain natural classes of ideals, where one can obtain much better complexity bounds.

2.1. **Polynomial systems.** Given polynomials

$$f_1, \ldots, f_m \in \mathbb{C}[x_1, \ldots, x_n],$$

we define their zero set by

$$Z(f_1, \ldots, f_m) \coloneqq \{\alpha \in \mathbb{C}^n \mid f_i(\alpha) = 0,\ \forall i\}.$$

A first basic question is:

$$Z(f_1, \ldots, f_m) = \varnothing?$$

This is the polynomial system satisfiability problem.

It is already computationally difficult. It is NP-hard, since Boolean satisfiability problems such as 3-SAT can be encoded as polynomial systems. On the other hand, by work of Koiran, assuming GRH, polynomial system satisfiability over $\mathbb{C}$ has upper bounds in the polynomial hierarchy.

2.2. **Irreducibility of zero sets.** The next geometric question is irreducibility.

Every algebraic variety Z can be decomposed as a finite union of irreducible varieties:

$$Z = \bigcup_i Z_i.$$

So one may ask:

Computationally, how can we find this decomposition?

Equivalently, for an ideal I, we can ask whether the variety

$$Z(I)$$

is irreducible.

This is closely related to primality of ideals. Geometrically, an ideal being prime corresponds to its zero set being irreducible, at least after passing to the right algebraic setting.

Thus, the problem of testing whether an ideal is prime is a vast generalization of testing whether a polynomial has irreducible zero set.

For a principal ideal

$$I = (f),$$

this becomes the classical problem of testing whether the polynomial f is absolutely irreducible.

For ideals with several generators, the problem becomes much harder.

2.3. **The complexity-theoretic picture.** The general primality testing problem for ideals has very high complexity. Previously, the best known upper bounds were around PSPACE or even EXPSPACE, depending on the input model and generality.

The talk discussed this problem in the context of the following rough "tower of complexity":

$$\text{EXPSPACE} \longrightarrow \text{PSPACE} \longrightarrow \text{PH} \longrightarrow \Sigma_2^p = NP^{NP} \longrightarrow NP \longrightarrow \text{practical/heuristic algorithms.}$$

Within this workshop, the discussion stays within decidable problems. Undecidable problems are not the focus here.

2.4. **Main result.** The main result of the talk is that, assuming the Generalized Riemann Hypothesis, primality testing for certain natural classes of ideals can be placed much lower in the complexity hierarchy.

More precisely, for radical ideals and complete intersection ideals, testing primality lies in the third level of the polynomial hierarchy.

This is almost optimal in the sense that these problems are already NP-hard.

Previously, the best known upper bound for these problems was PSPACE.

2.5. **Why zeros help.** The key idea is to study algebraic properties of ideals through their zero sets, and then understand how these properties behave after reducing coefficients modulo primes p.

The method generalizes Koiran's approach to polynomial system satisfiability. Instead of only asking whether a system has a common zero, one asks more refined geometric questions:

- Is the zero set irreducible?
- What is its dimension?
- Does the ideal remain prime after reduction modulo p?
- How do these properties behave for many primes p?

This connects computational complexity with effective algebraic geometry and commutative algebra.

2.6. **Takeaway.** The main takeaway is that primality testing for ideals is a geometric generalization of several classical computational problems:

- polynomial system satisfiability;
- absolute irreducibility testing of polynomials;
- irreducibility of algebraic varieties;
- primality of ideals.

The title *Primes via zeros* reflects this philosophy: instead of studying primality only algebraically through generators of an ideal, one studies the geometry of the corresponding zero set.

3. Speaker 3: Graph Isomorphism

Pascal Schweitzer, Technical University of Darmstadt
Tensor isomorphism's older, smaller, and tenacious sibling: the graph isomorphism problem

From the perspective of tensor isomorphism, graph isomorphism appears as a special case. It is the isomorphism problem for combinatorial objects given in an explicit, "verbose" representation. Tensor isomorphism, by contrast, often deals with more succinct algebraic representations, such as generators of matrix spaces or tensors.

Pascal Schweitzer's background is in graph isomorphism, structural and algorithmic graph theory, and certifying algorithms. In this talk, he gave an introduction to graph isomorphism and then explained its connections to logic, games, and the Weisfeiler–Leman algorithm.

3.1. **Graph isomorphism.** Two graphs G and H are isomorphic if there is a bijection

$$\varphi : V(G) \to V(H)$$

such that adjacency and non-adjacency are preserved. That is,

$$\{u, v\} \in E(G) \quad \iff \quad \{\varphi(u), \varphi(v)\} \in E(H).$$

An automorphism of a graph G is an isomorphism from G to itself. The automorphisms of G form a group, denoted

$$\mathrm{Aut}(G).$$

Theorem 3.1 (Mathon–Miller)**.** *The computation of automorphisms of arbitrary explicitly given combinatorial objects reduces to symmetry computation of graphs.*

This means that graph isomorphism and graph automorphism are central problems for understanding symmetries of finite combinatorial structures.

Theorem 3.2. *The following problems are polynomial-time Turing equivalent:*

- *the graph isomorphism problem;*
- *computing* $|\mathrm{Aut}(G)|$*;*
- *computing a generating set for* $\mathrm{Aut}(G)$*.*

So deciding whether two graphs are isomorphic is closely related to computing the full symmetry group of a graph.

3.2. **Graph isomorphism and other isomorphism problems.** Many other isomorphism problems can be reduced to graph isomorphism. For example, group isomorphism can be reduced to graph isomorphism by encoding the multiplication table or the relevant algebraic structure as a graph.

Thus, graph isomorphism acts as a kind of universal problem for explicitly represented finite combinatorial objects.

This is one reason it appears naturally in a workshop on tensor isomorphism: tensor isomorphism can be viewed as a more algebraic, more succinct analogue of graph isomorphism.

3.3. **Complexity of graph isomorphism.** The graph isomorphism problem is not known to be solvable in polynomial time, but it is also not known to be NP-complete.

The best general algorithm is Babai's quasipolynomial-time algorithm, with running time of the form

$$n^{O((\log n)^c)}$$

for some constant c.

Babai's algorithm is strongly connected to the string isomorphism problem. A classical motivating example is solving Rubik's cube, where one studies configurations under a group of allowed permutations.

Permutation groups are combinatorial objects too, and algorithms for permutation groups play a major role in graph isomorphism.

3.4. **Implicit versus explicit representations.** An important distinction is between implicit and explicit representations.

Graphs are explicit objects: one can list their vertices and edges.

Tensors, matrix spaces, and groups can be much more succinctly represented. For example, a matrix space may be given by a small list of generators, even though the full space may be very large.

This contrast is one of the conceptual differences between graph isomorphism and tensor isomorphism.

At this point, the introduction to graph isomorphism was complete.

3.5. **Logics and games.** The second part of the talk moved toward logic and games.

One can write logical formulas about graphs. For example, the following sentence says that there is a vertex which is either adjacent to every other vertex or adjacent to no other vertex:

$$\exists x \Big(\forall y \, (y \neq x \to E(x,y)) \ \vee \ \forall y \, (y \neq x \to \neg E(x,y)) \Big).$$

Here $E(x,y)$ means that x and y are adjacent.

The general idea is:

> If two graphs satisfy different logical sentences, then they cannot be isomorphic.

So logical formulas can be used as isomorphism tests.

3.6. **The Weisfeiler–Leman algorithm.** One important approach is the k-dimensional Weisfeiler–Leman algorithm, abbreviated k-WL.

The k-WL algorithm is a combinatorial algorithm which, given graphs G_1 and G_2,

- colors all k-tuples of vertices;
- iteratively refines these colors using the local structure of the graph;
- compares the resulting color patterns for the two graphs.

If the color patterns differ, then the two graphs are not isomorphic.

If the color patterns agree, then the graphs may still be non-isomorphic. Thus, k-WL gives a powerful but generally incomplete isomorphism test.

The case $k = 1$ is the color refinement algorithm. It colors vertices by their degrees and then repeatedly refines the coloring using the colors of neighboring vertices.

Higher-dimensional Weisfeiler–Leman algorithms use tuples of vertices instead of single vertices.

3.7. **Logic and Weisfeiler–Leman.** The Weisfeiler–Leman algorithm has a precise connection to finite-variable logic with counting quantifiers.

The relevant logic uses:

- a bounded number of variables;

- counting quantifiers.

For example, one can express statements such as:

> There is a vertex with at least 3 neighbors of degree at least 5.

The k-WL algorithm checks all invariants expressible in the corresponding counting logic with a bounded number of variables.

So the algorithmic question about graph isomorphism becomes related to the logical question:

> What kinds of graph properties can be expressed using only k variables?

3.8. **Pebble games.** Pebble games provide another way to understand the same phenomenon.

In a pebble game, two players place pebbles on vertices of two graphs. One player tries to show that the graphs are different, while the other tries to maintain the appearance of an isomorphism.

These games give a combinatorial interpretation of logical indistinguishability. If one player has a winning strategy, this corresponds to the existence of a logical formula that distinguishes the two graphs.

Thus, graph isomorphism, logic, the Weisfeiler–Leman algorithm, and pebble games are all different perspectives on the same underlying question.

3.9. **Main question.** The general question is:

> How complicated does a logical formula have to be in order to distinguish two graphs?

More specifically, one can ask:

- How many variables are needed?
- How many iterations of the Weisfeiler–Leman algorithm are needed?
- What graph classes are distinguishable by low-dimensional WL?
- What examples are hard for WL?

3.10. **Connection to groups.** The talk also emphasized that the behavior of Weisfeiler–Leman methods on groups is still not fully understood.

For graphs, the relationship between WL, logic, and pebble games is well developed. For groups, analogous methods exist, but their power and limitations are more subtle.

This is especially relevant for the workshop because group isomorphism and tensor isomorphism are deeply connected. In particular, some group isomorphism problems can be translated into tensor or bilinear-map isomorphism problems.

3.11. **Takeaway.** The main takeaway is that graph isomorphism is the classical model problem for isomorphism testing. It is smaller and more explicit than tensor isomorphism, but it already contains deep algorithmic, logical, and group-theoretic difficulties.

4. Speaker 4: Lattice Reduction

Antoine Joux, CISPA Helmholtz Center for Information Security

Indefiniteness makes lattice reduction easier

Antoine Joux is a cryptographer whose work focuses especially on cryptanalysis and on the algorithmic study of mathematical problems underlying public-key cryptography. This background explains the motivation of the talk: lattice reduction is one of the central algorithmic tools in computational number theory and cryptography.

The talk was about lattice reduction, starting from the classical LLL algorithm and then moving to indefinite lattices.

4.1. **Lattices and lattice isomorphism.** A lattice is a discrete subgroup of $\mathbb{R}^n$ equipped with a scalar product. Usually, we describe a lattice by giving a basis, or more generally a list of generating vectors.

If

$$L = (v_1, \ldots, v_d)$$

is a basis matrix whose columns are the basis vectors, then the Gram matrix of the lattice is

$$G_L = L^t L$$

when the ambient scalar product is the standard Euclidean one.

More generally, if the ambient space has scalar product given by a symmetric matrix G, then the Gram matrix of the lattice basis is

$$G_L = L^t G L.$$

Changing the lattice basis corresponds to multiplying by an integral invertible matrix. Thus, $GL_d(\mathbb{Z})$ acts on Gram matrices by

$$G_L \mapsto U^t G_L U, \qquad U \in GL_d(\mathbb{Z}).$$

If one restricts to orientation-preserving basis changes, one can use $SL_d(\mathbb{Z})$ instead.

The corresponding decision problem is lattice isomorphism: decide whether two lattices are isomorphic, equivalently whether their Gram matrices are equivalent under such an integral change of basis.

4.2. **What is LLL?.** LLL stands for the Lenstra–Lenstra–Lovasz algorithm, introduced in 1982.

It is a polynomial-time algorithm for lattice basis reduction. The goal is to start with an arbitrary basis of a lattice and produce a "better" basis.

What does "better" mean? There is no single perfect definition of the best basis of a lattice. Roughly, we want basis vectors that are short and close to orthogonal.

The LLL algorithm gives a polynomial-time method that produces a reasonably good basis. However, the quality guarantee becomes worse as the dimension grows.

4.3. **Specifying a lattice.** There are two common ways to specify a lattice.

First, one can give a list of generating vectors and use the standard scalar product implicitly.

Second, one can give the Gram matrix explicitly. In this viewpoint, the basis vectors are implicit, and all the metric information is encoded in the matrix.

The Gram matrix remembers the pairwise inner products:

$$(G_L)_{ij} = \langle v_i, v_j \rangle.$$

4.4. **Basic invariants.** Some basic invariants of a lattice are:

- the dimension of the lattice, which is the number of vectors in a basis;
- the determinant, or covolume, of the lattice.

If G_L is the Gram matrix, then the determinant of the lattice is

$$D_L = \sqrt{\det(G_L)}.$$

This is the volume of the fundamental parallelepiped spanned by the basis vectors.

4.5. **Main guarantee of LLL.** One of the main guarantees of the LLL algorithm is that the first vector v_1 in the reduced basis satisfies a bound of the form

$$\|v_1\| \leq \gamma^{d_L - 1} D_L^{1/d_L},$$

where d_L is the lattice dimension and

$$\gamma = (4/3)^{1/4} + \varepsilon.$$

Thus, LLL finds a vector whose length is controlled by the determinant of the lattice, but with an approximation factor that is exponential in the dimension.

This is still very useful in practice, but the dimension dependence is important.

4.6. **Main ingredients of LLL.** The main ingredients of the LLL algorithm are:

- Gram–Schmidt orthogonalization;
- size reduction;
- Gauss reduction on projected 2×2 lattices.

The algorithm repeatedly improves the basis using local two-dimensional reductions and orthogonalization data.

4.7. **Indefinite lattices.** The new part of the talk was about indefinite lattices.

Instead of a positive definite scalar product, take an arbitrary symmetric matrix G. Then the Gram matrix is still

$$G_L = L^t G L,$$

but G_L is no longer necessarily positive definite.

It may have positive, negative, and zero eigenvalues. In other words, the quadratic form may be indefinite or degenerate.

One can diagonalize the form and write it, after a suitable change of basis, in a block form such as

$$G_L = \begin{bmatrix} G_\ell & 0 \\ 0 & 0 \end{bmatrix},$$

where G_ℓ has only non-zero eigenvalues.

The signature of the form records the numbers of positive and negative eigenvalues.

4.8. **Generalizing LLL.** The classical LLL algorithm is designed for positive definite lattices. The question is whether one can generalize LLL to the indefinite case, where the scalar product is replaced by an arbitrary quadratic form.

This question had been considered before in work of Gabor Ivanyos and Agnes Szanto, and also independently by Denis Simon. These works generalized LLL to indefinite lattices and obtained polynomial-time algorithms with approximation factors analogous to the classical LLL setting.

The key point of Joux's talk was that indefinite lattices can behave better than expected.

Instead of an approximation factor depending exponentially on the full dimension, one can hope for a bound depending on the signature of the indefinite lattice.

This is the meaning of the title:

> Indefiniteness makes lattice reduction easier.

4.9. **Takeaway.** The classical LLL algorithm works for positive definite lattices and gives a polynomial-time reduction algorithm with approximation quality exponential in the dimension.

For indefinite lattices, the situation is more subtle. The quadratic form has positive and negative directions, and these directions can be exploited algorithmically.

The main takeaway is that indefinite lattice reduction may lead to much better reduced representations than previously thought. In particular, the relevant approximation factor can depend on the signature rather than on the full dimension.

5. Speaker 5: Continuous HSP

Fang Song, Portland State University
Continuous hidden subgroup problem

Fang Song works at the intersection of cryptography, quantum computing, and computational complexity. This background is very relevant for the talk, because the hidden subgroup problem is one of the central frameworks for understanding quantum algorithms and their applications to number theory and cryptography.

The talk introduced the continuous hidden subgroup problem, or continuous HSP. This is an extension of the usual hidden subgroup problem from finite groups to certain continuous groups, especially groups such as $\mathbb{R}^m$. The talk also explained how this framework leads to efficient quantum algorithms for basic problems in algebraic number theory.

5.1. **The hidden subgroup problem.** HSP stands for the hidden subgroup problem.

In the basic setting, we are given a group G, a set S, and a black-box function

$$f : G \to S.$$

The promise is that there exists a subgroup

$$H \leq G$$

such that f is constant on cosets of H and different on different cosets.

For additive groups, this can be written as:

$$f(x) = f(y) \quad \Longleftrightarrow \quad x - y \in H.$$

The goal is to find the hidden subgroup H using as few calls to f as possible.

Equivalently, if the cosets of H are

$$y_i + H,$$

then the function maps each coset to a distinct value:

$$y_i + H \mapsto s_i, \qquad s_i \neq s_j \text{ for } i \neq j.$$

So the function hides the subgroup H by revealing only which coset a point belongs to, not the subgroup directly.

5.2. **Spelling out the HSP properties.** The two main properties are:

- **Periodicity:** if $x \in y + H$, then
$$f(x) = f(y).$$
- **Injectivity on cosets:** if $x \notin y + H$, then
$$f(x) \neq f(y).$$

Thus, the function is periodic with respect to the subgroup H, and it separates different cosets of H.

5.3. **Examples and motivation.** The hidden subgroup problem is useful because many important computational problems can be reduced to HSP instances.

In the abelian case, we have examples such as:

- discrete logarithm reduces to HSP over
$$\mathbb{Z}_N \times \mathbb{Z}_N;$$
- factoring reduces to an HSP-type problem over
$$\mathbb{Z};$$

- breaking certain block-cipher structures can be related to HSP over
$$\mathbb{Z}_2^n;$$
- unit group computation, the principal ideal problem, and class group computation can be related to HSP over
$$\mathbb{R}^{\text{const}}$$
in the constant-degree number field case.

There are also important non-abelian examples:

- graph isomorphism can be reduced to HSP over a symmetric group;
- the unique shortest vector problem can be related to HSP over a dihedral group.

For finite abelian groups, HSP has efficient quantum algorithms. This generalizes the ideas behind Shor's algorithms for factoring and discrete logarithms.

For non-abelian groups, the situation is much more subtle. Some non-abelian HSPs have efficient quantum algorithms, but the general problem remains open.

5.4. **Quantum HSP: overview.** The quantum HSP algorithm starts from an information-theoretic point of view.

For example, take
$$G = \mathbb{Z}_{12}$$
and
$$H = \{0, 3, 6, 9\}.$$
The cosets are:
$$H, \quad 1 + H, \quad 2 + H.$$
A function hiding H is constant on each of these cosets and takes different values on different cosets.

The quantum algorithm creates a superposition over group elements, queries the oracle, and then obtains a coset state. The Fourier transform is then used to move into the frequency domain, where the hidden periodicity becomes easier to detect.

In this example, the relevant Fourier transform is the Fourier transform over
$$\mathbb{Z}_{12}.$$
More generally, Fourier analysis over finite abelian groups is the main tool for solving abelian HSPs efficiently on a quantum computer.

Question: Understand explicitly how the Fourier transform works for the example
$$G = \mathbb{Z}_{12}, \qquad H = \{0, 3, 6, 9\}.$$

Question: Visualize periodic discontinuous functions which are constant on cosets.

The useful part of the algorithm is that it translates the hidden-period problem into Fourier space, where the subgroup information becomes visible as orthogonality conditions.

One question to understand better is:

> How do we sample in the Fourier world on a quantum computer?

5.5. **Continuous HSP.** The continuous hidden subgroup problem extends this framework to continuous groups.

Instead of a finite group, consider
$$\mathbb{R}^m.$$
We are given a function
$$f : \mathbb{R}^m \to \mathcal{H},$$
where $\mathcal{H}$ is a Hilbert space of quantum states.

The hidden subgroup is now a full-rank lattice

$$L \subset \mathbb{R}^m.$$

There are some geometric conditions on the lattice, for example:

- the shortest vector of the lattice is not too short;
- the fundamental volume of the lattice is not too large.

These assumptions are needed so that the lattice can be recovered efficiently and robustly.

5.6. **The continuous oracle.** The function f is an (a, r, ε)-oracle if it satisfies the following properties.

- **Periodicity on L:**
$$x - y \in L \quad \Longrightarrow \quad |f(x)\rangle = |f(y)\rangle.$$
- **Pseudo-injectivity:** if x and y are far from being in the same coset of L, meaning
$$\min_{v \in L} \|x - y - v\| \geq r,$$
then
$$|\langle f(x) \mid f(y)\rangle| \leq \varepsilon.$$
In words, points that are far from the same lattice coset are mapped to almost orthogonal quantum states.
- **Lipschitz continuity:**
$$\big\| \, |f(x)\rangle - |f(y)\rangle \, \big\| \leq a\|x - y\|.$$
This says that nearby inputs are mapped to nearby quantum states.

These conditions replace the exact injectivity and exact discreteness of the finite HSP setting.

5.7. **Why continuity matters.** A difficulty in passing from finite groups to $\mathbb{R}^m$ is that digital or quantum computers cannot query an arbitrary real number with infinite precision.

If we simply restrict a function

$$f : \mathbb{R}^m \to S$$

to a discrete grid, we may lose the exact periodicity properties. This makes the naive discretization approach problematic, especially in high dimensions.

The continuous HSP framework handles this by allowing approximate behavior:

- exact periodicity with respect to the lattice;
- approximate orthogonality away from the lattice;
- Lipschitz continuity to control nearby points.

This makes the problem robust enough to support an efficient quantum algorithm.

5.8. **Applications.** The continuous HSP framework is important because it gives quantum algorithms for algebraic number theory problems.

Examples include:

- computing the unit group of a number field;
- the principal ideal problem;
- class group computation;
- related problems involving number fields and ideal lattices.

In earlier work, some of these problems could be treated efficiently only for number fields of constant degree. The continuous HSP framework is designed to handle higher-dimensional settings, such as

$$\mathbb{R}^{O(n)}$$

for a number field of degree n.

5.9. **Connection to cryptography.** The cryptographic motivation is that quantum algorithms can break many classical cryptographic systems. Shor's algorithm breaks systems based on factoring and discrete logarithms.

The continuous HSP framework raises a broader question:

> Which other algebraic and number-theoretic problems admit efficient quantum algorithms?

This is especially relevant for cryptographic systems related to number fields and ideal lattices.

5.10. **Takeaway.** The main takeaway is that HSP is a unifying framework for quantum algorithms. In the finite abelian case, it explains algorithms such as Shor's factoring and discrete logarithm algorithms.

The continuous HSP extends this framework to groups such as

$$\mathbb{R}^m,$$

where the hidden subgroup is a lattice.

Notes by Maksym Zubkov