The Graph Isomorphism Problem

MARTIN GROHE, RWTH Aachen University, Germany
PASCAL SCHWEITZER,
Technical University of Kaiserslautern, Germany

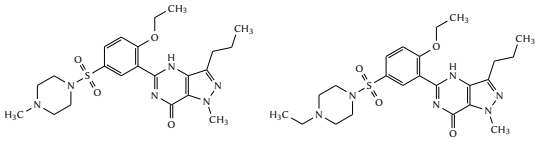


Fig. 1. Non-isomorphic molecular graphs

ACM Reference Format:
Martin Grohe and Pascal Schweitzer. 2018. The Graph Isomorphism Problem. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Deciding whether two graphs are structurally identical, or *isomorphic*, is a classical algorithmic problem that has been studied since the early days of computing. Applications span a broad field of areas ranging from chemistry (cf. Figure 1) to computer vision. Closely related is the problem of detecting symmetries of graphs and of general combinatorial structures. Again this has many application domains, for example combinatorial optimisation, the generation of combinatorial structures, and the computation of normal forms. On the more theoretical side, the problem is of central interest in areas such as logic, algorithmic group theory, and quantum computing.

Graph isomorphism (GI) gained prominence in the theory community in the 1970s, when it emerged as one of the few natural problems in the complexity class NP that could neither be classified as being hard (NP-complete) nor shown to be solvable with an efficient algorithm (i.e., have a polynomial-time algorithm). It was mentioned numerous times as an open problem, in particular already in Karp's seminal 1972 paper [23] on NP-completeness as well as in Garey and Johnson's influential book on computers and intractability [15]. Since then, determining the precise computational complexity of GI has been regarded a major open problem in theoretical computer science.

In a recent breakthrough [3], Babai proved that GI is solvable in *quasipolynomial time*. This means that on n-vertex input graphs, Babai's algorithm runs in time $n^{p(\log n)}$ for some polynomial $p(X)$. This can be interpreted as the problem being almost efficiently solvable – theoretically.

In this article, we will survey both theoretical and practical aspects of the graph isomorphism problem, paying particular attention to the developments that led to Babai's result.

Historical Development

Graph isomorphism as a computational problem first appears in the chemical documentation literature of the 1950s (e.g. [35]) as the problem of matching a molecular graph (cf. Figure 1) against a database of such graphs. The earliest computer science reference we are aware of is due to Unger [39], incidentally also in the *Communications of the ACM*. Maybe the first important step on the theoretical side was Hopcroft and Tarjan's $O(n \log n)$ isomorphism algorithm for planar graphs [22]. As the question whether GI is NP-complete gained prominence, it was realised that GI has aspects that distinguish it from most NP-complete problems. In particular, counting the number of isomorphisms between two graphs is not harder than deciding if there is an isomorphism (see [29]). Babai, Erdös and Selkow [8] showed that GI is easy on average with respect to a

Manuscript submitted to ACM

uniform distribution of input graphs. In fact, this can be extended to most other random graph distributions.

A first wave of substantial progress came in 1979/80 with Babai [2] and Luks's [26] introduction of group theoretic techniques. In his paper [26] showing that isomorphism can be decided in polynomial time on graph classes of bounded degree (i.e., the number of edges incident with each vertex is bounded), Luks laid the foundation for much of the subsequent work on graph isomorphism algorithms by introducing a general divide-and-conquer framework. (We will discuss this framework in some detail in Section 3.) A combination of Luks's group theoretic framework with a clever combinatorial trick by Zemlyachenko led to a moderately exponential algorithm for graph isomorphism (see [10]). The best bound was $2^{O(\sqrt{n \log n})}$, established by Luks in 1983 (see [9]). This remained the best known bound until Babai's recent breakthrough.

Around the same time, McKay developed his isomorphism tool Nauty [30], which marks a breakthrough in practical isomorphism testing (see Section 4).

In the mid-1980s, another fascinating facet of the complexity of GI was discovered. Using the newly developed machinery of interactive proof systems, it was shown that the complement of GI has short zero knowledge proofs [16] and this was seen as another indication that GI is not NP-complete. (If GI is NP-complete, then the so-called polynomial hierarchy of complexity classes above NP collapses to its second-level, which is regarded as unlikely.) On the other hand Torán [38] proved that GI is hard to solve when the available memory is quite limited (specifically it is hard for nondeterministic logarithmic space under logspace reductions), which gives us at least some complexity theoretic lower bound.

Since the group theoretic methods have been introduced in the early 1980s, they have been continually refined. The underlying group theory has progressed (e.g. [5, 9]), the complexity of the group theoretic problems has been analysed in detail (e.g. [27, 37]), and the scope of the methods has been expanded to other structures (e.g. [7]). Another active strand of research has been the design of efficient

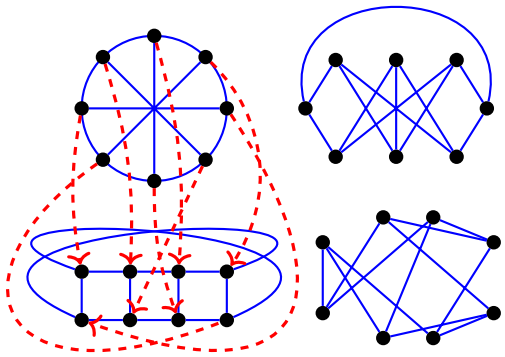


Fig. 2. Four isomorphic graphs. The red arrows indicate an isomorphism between the first and the third graph.

algorithms for GI restricted to graphs with specific properties (e.g. [6, 19, 25, 34]). More recently, there has also been work on memory restricted algorithms (e.g. [14]).

But no real progress on the general isomorphism problem was made until — out of the blue — Babai published his quasipolynomial-time algorithm in 2015.

After this historical overview, let us get slightly more concrete.

Isomorphisms, Automorphisms, and Canonical Forms

An *isomorphism* from a graph $G = (V, E)$ to a graph $H = (W, F)$ is a one-to-one mapping π from the vertices of the first graph V onto the vertices of the second graph W that preserves adjacency and non-adjacency, that is, $uv \in E$ if and only if $\pi(u)\pi(v) \in F$ for all pairs uv of vertices in V (cf. Figure 2).

An *automorphism*, or a symmetry, of a graph G is an isomorphism from G to G itself. For example, all $n!$ permutations of the vertex set of a complete graph K_n on n vertices are automorphisms. By comparison, an (undirected) path of length n only has two automorphisms, the trivial identity mapping, and the mapping that flips the ends of the path. The collection of all automorphisms of G forms a mathematical structure known as a (permutation) group. As the example of the complete graph shows, automorphism groups can get very large, exponentially

large in the number of vertices, but fortunately every permutation group has a generating set linear in the size of the permutation domain (i.e., the set of objects being permuted). This allows us to work with automorphism groups efficiently as long as they are represented by sufficiently small generating sets. The problem GI of deciding whether two graphs are isomorphic and the problem AUT of computing a generating set for the automorphism group of a graph have the same computational complexity, or more precisely, can be reduced to each other by polynomial-time reductions (see [29]).

Another important related problem is the graph canonisation problem. A *canonical form* γ maps each graph G to an isomorphic graph $\gamma(G)$ in such a way that if graphs G and H are isomorphic then the graphs $\gamma(G)$ and $\gamma(H)$ are identical (not just isomorphic). Observe that a canonical form γ yields an isomorphism test: given G, H, compute $\gamma(G)$ and $\gamma(H)$ and check if they are identical. In practical applications, canonical forms are often preferable over isomorphism tests. It is an open problem whether these two problems are actually equivalent (e.g., whether the existence of a polynomial-time isomorphism algorithm would yield the existence of a polynomial-time computable canonical form). However, typically for graph classes for which we know a polynomial-time isomorphism algorithm we also have a polynomial-time canonisation algorithm. Sometimes, the extension from isomorphism testing to canonisation is straightforward; sometimes it requires extra work (e.g., [4, 10, 36]).

2 COMBINATORIAL ALGORITHMS

To establish that two graphs are isomorphic, we can try to find an isomorphism. To establish that the graphs are non-isomorphic, we can try to find a "certificate" of non-isomorphism. For example, we can count vertices, edges, triangles in both graphs; if any of these counts differ, the graphs are non-isomorphic. Or we can look at the degrees of the vertices. If there is some d such that the two graphs have a different number of vertices of degree d, the graphs are non-isomorphic.

The *Weisfeiler-Leman algorithm* provides a systematic approach to generating such certificates of non-isomorphism in an efficient way. Actually, it is a whole family of algorithms, parameterised by a positive integer, the *dimension*.

Colour Refinement

We start by describing the 1-dimensional version, which is commonly known as *colour refinement* or *naive vertex classification*. It is one of the most basic ideas in graph isomorphism testing that has been re-invented several times; the oldest published version that we are aware of can be found in [32]. Colour refinement is an important subroutine of almost all practical graph isomorphism tools (cf. Section 4), and it is also a building block for many theoretical results.

Colour Refinement

Input: Graph G
Initialisation: All vertices get the same colour.
Refinement Step: For all colours c in the current colouring and all nodes v, w of colour c, v and w get different colours in the new colouring if there is some colour d such that v and w have different numbers of neighbours of colour d.

The refinement is repeated until the colouring is stable, then the stable colouring is returned.

Fig. 3. The colour refinement algorithm

The colour refinement algorithm, displayed in Figure 3, iteratively computes a colouring of the vertices of a graph. The actual colours used are irrelevant, what matters is the partition of the vertices into colour classes. The final colouring has the property that any two vertices of the same colour have the same number of neighbours in each colour class. Figure 4 shows an example.

The colouring computed by the algorithm is *isomorphism invariant*, which means that if we run it on two isomorphic graphs the resulting coloured graphs will still be isomorphic and in particular have the same numbers of nodes of each colour. Thus, if we run the algorithm on two graphs and find that they have distinct numbers of vertices of some colour, we have produced a certificate of

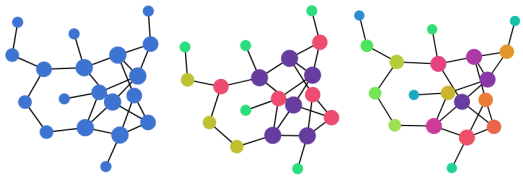


Fig. 4. Colour refinement: a graph, its colouring after 1 refinement round, and the final colouring

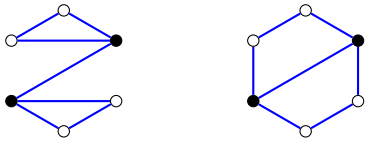


Fig. 5. Two non-isomorphic graphs that are not distinguished by colour refinement. Colour-refinement computes the black-white colouring of the vertices.

non-isomorphism. If this is the case, we say that colour refinement *distinguishes* the two graphs.

Unfortunately, colour refinement does not distinguish all non-isomorphic graphs. Figure 5 shows a simple example. But, remarkably, colour refinement does distinguish *almost all* graphs, in a precise probabilistic sense [8]. This, together with its efficiency, is what makes colour refinement so useful as a subroutine of practical isomorphism tools.

The reader may have noticed that colour refinement is very similar to other partitioning algorithms, in particular the standard algorithm for minimising deterministic finite automata. Borrowing ideas from Hopcroft's DFA minimisation algorithm [21], colour refinement can be implemented to run in time $O((n+m)\log n)$, where n is the number of vertices and m the number of edges of the input graph [13]. Thus colour refinement is indeed very efficient.

Weisfeiler-Leman

We have seen that colour refinement is not a complete isomorphism test: it fails to distinguish extremely simple non-isomorphic graphs such as those shown in Figure 5. The *k-dimensional Weisfeiler-Leman algorithm (k-WL)* is based on the same iterative-refinement idea as colour refinement, but is significantly more powerful. Instead of vertices, k-WL colours k-tuples of vertices of a graph. Initially, each k-tuple is "coloured" by the isomorphism type of the subgraph it induces. Then in the refinement rounds, the colour information is propagated between "adjacent" tuples that only differ in one coordinate (details can be found in [11]). The 2-dimensional version of the algorithm is due to Weisfeiler and Leman [40]; the generalisation to higher dimensions is due to Faradžev, Zemlyachenko, Babai, and Mathon (see [11]). If implemented using similar ideas as for colour refinement, k-WL runs in time $O(n^{k+1}\log n)$.

Higher-dimensional WL is very powerful. Indeed, it is highly nontrivial to find non-isomorphic graphs that are not distinguished by 3-WL. It took a now seminal paper, by Cai, Fürer, and Immerman [11], to prove that for every k there are non-isomorphic graphs G_k, H_k that are not distinguished by k-WL. Indeed, these graphs, known as the *CFI graphs*, have size $O(k)$ and are 3-regular.

It turns out that many natural graph classes do not admit the CFI-graph construction and a low-dimensional WL is a complete isomorphism test. In particular, for all graph classes C that exclude some fixed graph as a minor, there is a constant k such that k-WL distinguishes all non-isomorphic graphs in C [17]. This includes the class of planar graphs, for which 3-WL suffices [24].

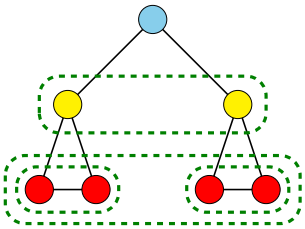
The Weisfeiler-Leman algorithm is remarkably robust. It not only subsumes most combinatorial ideas for graph isomorphism testing, but also has a natural characterisation in terms of logic [11]. Surprisingly, it also corresponds to a natural isomorphism test based on linear programming [1] and subsumes various approaches to GI based on algebraic and mathematical programming techniques.

3 GROUP THEORETIC ALGORITHMS

While most isomorphism algorithms devised over the years are subsumed by the Weisfeiler-Leman algorithm, this is not the case for the group theoretic approach [2, 11]. A first application of algorithmic group theory to isomorphism testing was given by Babai [2]. Subsequently Luks [26]

Basic permutation group concepts

Permutation Domain: The objects that are permuted
Symmetric group: all permutations
Alternating group: even permutations, i.e., products of an even number of transpositions
The giants: the symmetric and the alternating group
Orbits: equivalence classes of objects that can be mapped to each other
Transitive group: every object can be mapped to every other object, i.e., only one orbit
Block: A subset of the objects that is always mapped to itself or somewhere else entirely
Primitive group: permutation group whose blocks are singletons or the entire permutation domain.
Example:



Vertices that are in the same orbit of the automorphism group of the graph are coloured with the same colour. The sets bordered by dashed lines are examples of blocks.

Fig. 6. Basic Permutation group concepts

used a group theoretic approach to devise a polynomial-time isomorphism test for graphs of bounded degree.

Since GI and the automorphism group problem AUT are polynomially equivalent (see [29]), it suffices to solve the latter. Starting with a suitable group of permutations we want to compute within it the automorphism group A of interest (technically we want to compute a certain set-stabilizer or the solution to a string isomorphism problem, which we will not elaborate on here). We continually maintain an encasing group $\Gamma \geq A$ containing all automorphisms as a subgroup. Our strategy is to iteratively shrink Γ until it agrees with A.

To shrink the group Γ, in case the permutation group Γ has more than one orbit (see Figure 6), we process orbits sequentially.

If the group has only one orbit, we exploit so-called blocks whenever they exist. A *block* of a permutation group $\Gamma \leq \mathrm{Sym}(\Omega)$ is a subset always mapped to itself or somewhere else entirely, that is, a set $B \subseteq \Omega$ of the permutation domain Ω such that for all $\gamma \in \Gamma$ we have $\gamma(B) = B$ or $\gamma(B) \cap B = \{\}$. The set of images of the block $\{\gamma(B) \mid \gamma \in \Gamma\}$ partitions the domain Ω into blocks of equal size, which together form a so-called *block system*. The group Γ permutes the blocks of the system and we can consider the induced permutation group Γ' on the blocks. By choosing $B \subsetneq \Omega$ inclusion-wise maximal among the blocks, we can ensure that Γ' does not have any (non-trivial) blocks itself. A group with this property is called *primitive*. Luks argues that in polynomial time we can reduce the computation of the automorphism group A to $|\Gamma'|$ computations, each involving subproblems with significantly smaller orbits, which can then be processed sequentially as mentioned above. In case we started with a primitive group we use a brute force algorithm, inspecting all permutations in Γ separately.

A crucial observation is now that for graphs of bounded degree, there is a method to guarantee that $|\Gamma'|$ cannot be too large. Originally Luks presented a more involved argument but a subsequent result by Babai, Cameron and Pálfy [5] directly shows that $|\Gamma'|$ is polynomially bounded in the permutation domain size. Overall this bound implies that the entire procedure runs in polynomial time on graphs of bounded degree.

For general graphs, the bottleneck of this procedure occurs when Γ' is large. In that case Γ' is a large primitive group. Such groups are called *Cameron groups* and a precise classification is known [12, 28]. However, this is not a new insight and the fact that Cameron groups form the bottleneck to improving Luks's method was already known in the 1980s.

Babai's quasipolynomial-time algorithm

Attacking exactly this bottleneck, 35 years later it was Babai who improved the running time of the theoretically fastest general graph isomorphism algorithm. He showed that graph isomorphism can be solved in quasipolynomial

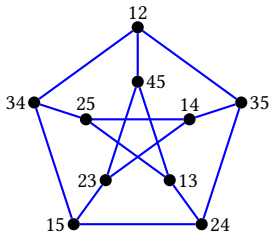


Fig. 7. The *Petersen graph* is a small graph whose automorphism is a Johnson group. Its nodes correspond to the 2-element subsets of $\{1,\dots,5\}$, with an edge between two nodes if the corresponding subsets have an empty intersection. The automorphism group of the Petersen graph is the symmetric group S_5 with its natural action on the 2-element subsets.

time $n^{\text{polylog}(n)}$, that is $n^{(\log(n)^c)}$ for some constant c. Doing his algorithmic ideas justice is difficult not only because they span 80 pages in his original manuscript but also because the algorithm contains several major, very distinct new ideas that combine smoothly to an overall algorithm. Here, we can thus only sketch the underlying ideas of the various puzzle pieces and how they are combined.

The first step, at quasipolynomial cost, is to reduce the bottleneck of Cameron groups further to what Babai calls *Johnson groups*. They are groups abstractly isomorphic to a symmetric or an alternating group, but do not necessarily act in their natural action of permuting elements in some ground set, and rather consist of permutations of the t-element subsets of the ground set. An example is the automorphism group of the *Petersen graph* (see Fig 7).

As next step, to make Luks's framework more flexible for his recursive algorithm, Babai does not only maintain an encasing group Γ containing the automorphism group, but also a homomorphism $\varphi\colon \Gamma \to \Gamma'$ into a permutation group $\Gamma' \leq \mathrm{Sym}(\Omega')$ over an *ideal domain* Ω'. This allows the algorithm to make progress by decreasing the size of the ideal domain.

Initially, for the Johnson groups, we can choose as ideal domain the abstract ground set mentioned above. This way, the image of φ contains almost all permutations, i.e., it is the symmetric group or the alternating group on the ideal domain. These two groups are by far the largest primitive groups and therefore called the *giants*. Accordingly we speak of a *giant homomorphism*.

The general strategy of the algorithm is to reduce a problem instance to quasipolynomially many instances that are all smaller than the original instance by at least a constant factor. This is continued until the recursive instances are sufficiently small to be resolved with brute force, leading to an overall quasipolynomial-time algorithm.

If the permutation group induced by the homomorphism φ on the ideal domain is intransitive or imprimitive, we can use the strategies of Luks to process orbits sequentially or to consider the actions on blocks, respectively. For this some non-trivial group theory is required to pull back information from the ideal domain to the original domain.

In case the subgroup A of Γ that we are interested in maps onto the alternating group of the ideal domain, computation of A is comparatively easy, so let us focus on the case that the image of the subgroup is not a giant.

We then use a *local to global* approach. We first collect local certificates by testing, for all logarithmic-size subsets T of the ideal domain, whether the homomorphism φ applied to the sought-after automorphism group A and restricted to T is a giant homomorphism. We call these sets *test sets*. A test set is *full* if the said restriction is a giant homomorphism. Since the test set size is only logarithmic and there are only quasipolynomially many such test sets, we can test all test sets for fullness using recursion.

If a test set is full, which certifies high local symmetry, there must be global symmetries certifying this and quite surprisingly such global symmetries can be efficiently constructed. At the core of this statement lies the *Unaffected Stabiliser Lemma*, a central insight proven by Babai.

If there are a lot of full test sets, the global symmetries allow for efficient recursion. On the other hand if only few test sets are full, the graph must have a nontrivial structural invariant. Furthermore, we can use the logarithmic-dimensional Weisfeiler-Leman algorithm to construct such a structural invariant in the form of a relational structure of logarithmic arity. This breaks the apparent symmetry.

With the *design lemma* we can reduce the relational structure of logarithmic arity to a structure with a binary

relation. We obtain a uniprimitive coherent configuration, a particular structure important in algebraic graph theory closely related to the 2-dimensional Weisfeiler-Leman algorithm.

The final puzzle piece is the *Split-Or-Johnson* combinatorial partitioning algorithm which from a uniprimitive coherent configuration either produces a split or finds a large canonically embedded Johnson graph, a graph whose automorphism group is a Johnson group. In fact splits, which are invariant partitions of the ideal domain akin to the blocks of a permutation group, can also occur during other parts of the algorithm. They are handled with the techniques similar to the imprimitive case of Luks's algorithm.

We are left with the case in which a large canonically embedded Johnson graph has been produced. After all, this case had to occur at some point, since we know that the resilient Johnson groups exist. But now the Johnson graph is in fact a blessing since we can exploit the well-understood structure of the Johnson graphs to dramatically decrease the size of the ideal domain.

Overall we obtain a quasipolynomial-time algorithm solving the general graph isomorphism problem. Besides the original manuscript, there is also detailed explanation of the algorithm in the Bourbaki series written by Helfgott (see [20] for an English translation). In fact, Helfgott detected an error in the Split-or-Johnson routine which however was quickly fixed by Babai.

Babai's algorithm depends on the classification of the finite simple groups, an enormous theorem spanning several hundred journal articles written by numerous authors. Many group theorists prefer to avoid the theorem and indeed Pyber modified Babai's algorithm to give an alternative analysis that does not depend on the classification.

In further advances Babai recently extended his result to a canonisation algorithm that runs in quasipolynomial time [4], and there is an improvement on Luks's original result for graphs of maximum degree d testing isomorphism in time $n^{(\log(d)^c)}$ [18].

4 PRACTICAL GRAPH ISOMORPHISM

In practice it is excessive to even run the 2-dimensional Weisfeiler-Leman algorithm, let alone some logarithmic-dimensional version as in Babai's algorithm. Current isomorphism packages rather use colour refinement, i.e., the 1-dimensional version. As mentioned, this is already sufficient for almost all graphs. If it turns out not to be sufficient, the algorithms take the route of branching by using the concept of individualisation.

Specifically, the *individualisation-refinement* paradigm, which is adopted by virtually all modern competitive isomorphism tools, one by one artificially assigns a different colour to all the vertices in a colour class. This breaks the symmetry and subsequently colour refinement can be potentially applied again to produce a more refined partition of the vertices. In a backtracking manner the tools continue until a discrete colour (a colouring in which all colour classes are a singleton) has been reached. The tools use various pruning techniques, such as invariants and pruning with automorphisms, discovered with intricate methods, to drastically improve their performance.

The tools actually compute a canonical form, which also solves the isomorphism problem (as explained at the end of Section 1). This highly practical method was originally pioneered by McKay with his famous software tool Nauty. There are now various extremely efficient packages such as Bliss, Conauto, Nauty, Saucy, and Traces freely available. Recently many new ideas, responsible for their efficiency, such as the use of the trace for early abortion of colour refinement in Traces, have found their way into the tools. We refer the reader to an extensive survey [31]. In contrast to Babai's quasipolynomial-time algorithm there are however graphs on which the running time of all individualisation-refinement algorithms scale exponentially [33].

5 APPLICATIONS

Graph isomorphism tools can in practice be used to find symmetries of combinatorial objects and as such they have numerous applications in miscellaneous domains. In the

context of optimisation, for example in SAT solving, symmetries are exploited to collapse the search space, since parts equivalent under symmetries only need to be explored once. An alternative way of exploiting symmetries is to add symmetry breaking constraints to the original input again drastically improving performance.

Another application domain exploits canonical labelling to store graph structured data in a data base. For example when molecules are stored in a chemical data base, the idea is to store only a canonical representative. To look up a given molecule in the data base, we compute its canonical representative and find the result in the data base. This way, no isomorphism tests against the elements in the data base are required. Other application domains include machine learning, computer graphics, software verification, model checking, and mathematical programming.

6 CONCLUDING REMARKS

With Babai's quasipolynomial time algorithm we have seen a breakthrough on one of the oldest and best studied algorithmic problems. Undoubtedly, this algorithm and its underlying mathematical framework rank among the most important contributions to theoretical computer science in a long time. We are only starting to explore the potential of the wealth of new ideas they bring to the field.

Current challenges include the group isomorphism problem, one of the core obstacles to even faster graph isomorphism tests. On the practical side, emerging applications in areas such as machine learning demand a better understanding of approximate versions of isomorphism and similarity measures between graphs.

Yet the question whether graph isomorphism is solvable in polynomial time remains open, and we can expect further deep, exiting insights until it will finally be settled.

REFERENCES

[1] A. Atserias and E. Maneva. 2013. Sherali–Adams Relaxations and Indistinguishability in Counting Logics. *SIAM J. Comput.* 42, 1 (2013), 112–137.

[2] L. Babai. 1979. *Monte Carlo algorithms in graph isomorphism testing.* Technical Report D.M.S. No. 79-10. Université de Montréal.

[3] L. Babai. 2016. Graph Isomorphism in Quasipolynomial Time. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC '16).* 684–697.

[4] L. Babai. 2019. Canonical form for graphs in quasipolynomial time. In *Proceedings of the 51th Annual ACM SIGACT Symposium on Theory of Computing.*

[5] L. Babai, P. J. Cameron, and P. P. Pálfy. 1982. On the orders of primitive groups with restricted nonabelian composition factors. *J. Algebra* 79, 1 (1982), 161–168. https://doi.org/10.1016/0021-8693(82)90323-4

[6] L. Babai, X. Chen, X. Sun, S.-H. Teng, and J. Wilmes. 2013. Faster Canonical Forms for Strongly Regular Graphs. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science.* 157–166.

[7] L. Babai, P. Codenotti, J. Grochow, and Y. Qiao. 2011. Code equivalence and group isomorphism. In *Proceedings of the 22nd annual ACM-SIAM Symposium on Discrete Algorithms.* 1395–1408.

[8] L. Babai, P. Erdös, and S. Selkow. 1980. Random graph isomorphism. *SIAM Journal on Computation* 9 (1980), 628–635.

[9] L. Babai, W.M. Kantor, and E.M. Luks. 1983. Computational Complexity and the Classification of Finite Simple Groups. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science.* 162–171.

[10] L. Babai and E.M. Luks. 1983. Canonical labeling of graphs. In *Proceedings of the 15th ACM Symposium on Theory of Computing.* 171–183.

[11] J. Cai, M. Fürer, and N. Immerman. 1992. An optimal lower bound on the number of variables for graph identification. *Combinatorica* 12 (1992), 389–410.

[12] P.J. Cameron. 1981. Finite permutation groups and finite simple groups. *Bulletin of the London Mathematical Society* 13, 1 (1981), 1–22.

[13] A. Cardon and M. Crochemore. 1982. Partitioning a graph in $O(|A| \log_2 |V|)$. *Theoretical Computer Science* 19, 1 (1982), 85 – 98.

[14] S. Datta, N. Limaye, P. Nimbhorkar, T. Thierauf, and F. Wagner. 2009. Planar Graph Isomorphism is in Log-Space. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity.* 203–214.

[15] M.R. Garey and D.S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman.

[16] O. Goldreich, S. Micali, and A. Wigderson. 1986. Proofs that Yield Nothing But their Validity and a Methodology of Cryptographic Protocol Design. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science.* 174–187.

[17] M. Grohe. 2017. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory.* Lecture Notes in Logic, Vol. 47. Cambridge University Press.

[18] M. Grohe, D.Neuen, and P. Schweitzer. 2018. A Faster Isomorphism Test for Graphs of Small Degree. In *Proceedings of the 59th Annual IEEE Symposium on Foundations of Computer Science.* 89–100.

[19] M. Grohe and D. Marx. 2015. Structure Theorem and Isomorphism Test for Graphs with Excluded Topological Subgraphs. *SIAM J. Comput.* 44, 1 (2015), 114–159.

[20] H.A. Helfgott, J. Bajpai, and D. Dona. 2017. Graph isomorphisms in quasi-polynomial time. *ArXiv* 1710.04574 (2017).

[21] J.E. Hopcroft. 1971. An n log n algorithm for minimizing states in a finite automaton. In *Theory of Machines and Computations*, Z. Kohavi and A. Paz (Eds.). Academic Press, 189–196.

[22] J.E. Hopcroft and R. Tarjan. 1972. Isomorphism of planar graphs (working paper). In *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher (Eds.). Plenum Press.

[23] R.M. Karp. 1972. Reducibilities among combinatorial problems. In *Complexity of Computer Computations*, R.E. Miller and J.W. Thatcher (Eds.). Plenum Press, New York, 85–103.

[24] S. Kiefer, I. Ponomarenko, and P. Schweitzer. 2017. The Weisfeiler-Leman dimension of planar graphs is at most 3. In *Proceedings of the 32nd ACM-IEEE Symposium on Logic in Computer Science*.

[25] D. Lokshtanov, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. 2014. Fixed-Parameter Tractable Canonization and Isomorphism Test for Graphs of Bounded Treewidth. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science*. 186–195.

[26] E.M. Luks. 1982. Isomorphism of Graphs of Bounded Valance Can Be Tested in Polynomial Time. *J. Comput. System Sci.* 25 (1982), 42–65.

[27] E.M. Luks. 1991. Permutation Groups and Polynomial-Time Computation. In *Groups And Computation, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, October 7-10, 1991 (DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 11)*, L. Finkelstein and W.M. Kantor (Eds.). DIMACS/AMS, 139–176.

[28] A. Maróti. 2002. On the orders of primitive groups. *Journal of Algebra* 258, 2 (2002), 631–640.

[29] R. Mathon. 1979. A Note on the Graph Isomorphism counting Problem. *Inform. Process. Lett.* 8, 3 (1979), 131–132.

[30] B. McKay. 1981. Practical graph isomorphism. *Congressus Numerantium* 30 (1981), 45–87.

[31] B.D. McKay and A. Piperno. 2014. Practical graph isomorphism, II. *Journal of Symbolic Compututation* 60 (2014), 94–112.

[32] H.L. Morgan. 1965. The generation of a unique machine description for chemical structures-a technique developed at chemical abstracts service. *Journal of Chemical Documentation* 5, 2 (1965), 107–113.

[33] D. Neuen and P. Schweitzer. 2018. An exponential lower bound for individualization-refinement algorithms for graph isomorphism. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*. 138–150.

[34] I. N. Ponomarenko. 1988. The isomorphism problem for classes of graphs that are invariant with respect to contraction. *Zap. Nauchn. Sem. Leningrad. Otdel. Mat. Inst. Steklov. (LOMI)* 174, Teor. Slozhn. Vychisl. 3 (1988), 147–177, 182. In Russian.

[35] L.C. Ray and R.A. Kirsch. 1957. Finding Chemical Records by Digital Computers. *Science* 126 (1957).

[36] P. Schweitzer and D. Wiebking. 2019. A unifying method for the design of algorithms canonizing combinatorial objects. In *Proceedings of the 51th Annual ACM SIGACT Symposium on Theory of Computing*. 1247–1258.

[37] Ákos Seress. 2003. *Permutation group algorithms*. Cambridge Tracts in Mathematics, Vol. 152. Cambridge University Press, Cambridge. x+264 pages. https://doi.org/10.1017/CBO9780511546549

[38] J. Torán. 2004. On the Hardness of Graph Isomorphism. *SIAM J. Comput.* 33, 5 (2004), 1093–1108.

[39] S. Unger. 1964. GIT—A Heuristic Program for Testing Pairs of Directed Line Graphs for Isomorphism. *Commun. ACM* 7, 1 (1964), 26–34.

[40] B. Weisfeiler and A. Leman. 1968. The reduction of a graph to canonical form and the algebgra which appears therein. *NTI, Series 2* (1968). English transalation by G. Ryabov avalable at https://www.iti.zcu.cz/wl2018/pdf/wl_paper_translation.pdf.