Isomorphism, Canonisation, and Definability for Graphs of Bounded Rank Width

Martin Grohe
RWTH Aachen University
Ahornstr. 55
Aachen, Germany
grohe@informatik.rwth-aachen.de

Daniel Neuen
RWTH Aachen University
Ahornstr. 55
Aachen, Germany
neuen@informatik.rwth-aachen.de

ABSTRACT

We investigate the interplay between the graph isomorphism problem, logical definability, and structural graph theory on a rich family of dense graph classes: graph classes of bounded rank width. We prove that the combinatorial Weisfeiler-Leman algorithm of dimension $(3k + 4)$ is a complete isomorphism test for the class of all graphs of rank width at most k. A consequence of our result is the first polynomial time canonisation algorithm for graphs of bounded rank width.

Our second main result addresses an open problem in descriptive complexity theory: we show that fixed-point logic with counting expresses precisely the polynomial time properties of graphs of bounded rank width.

1. INTRODUCTION

The question of whether there is an efficient algorithm deciding whether two graphs are isomorphic is one of the oldest an best-known open problems in theoretical computer science. Already mentioned in Karp's [18] seminal article on NP-complete combinatorial problems, graph isomorphism (from now on: GI) has remained one of the few natural problems in NP that is neither known to be solvable in polynomial time nor known to be NP-complete. The problem has received considerable attention recently because of Babai's [1] breakthrough algorithm deciding GI in quasipolynomial time $n^{\text{polylog}(n)}$.

However, the question of whether GI is solvable in polynomial time remains wide open. Polynomial time algorithms are known for the restrictions of GI to many interesting graph classes, for example the class of planar graphs [14], classes of bounded degree [19], even all classes excluding a fixed graph as a topological subgraph [9], and only recently, graph classes of bounded rank width [11].

Rank width, introduced by Oum and Seymour [21], is a graph invariant that measures how well a graph can be decomposed hierarchically in a certain style. In this respect, it is similar to the better-known tree width, but where tree width measures the complexity, or width, of a separation in a hierarchical decomposition in terms of the "connectivity" between the two sides, rank width measures the complexity of a separation in terms of the rank of the adjacency matrix of the edges between the two sides of the separation (see Figure 1). As opposed to tree width, rank width is also a meaningful measure of simplicity for dense graphs. For example, the rank width of a complete graph, arguably the simplest dense graph, is 1. Rank width is closely related to clique width [5], another well-studied graph invariant based on graph grammars. Many hard algorithmic problems can be solved efficiently on graphs of bounded rank width, or equivalently, bounded clique width, among them all problems definable in monadic second-order logic [4].

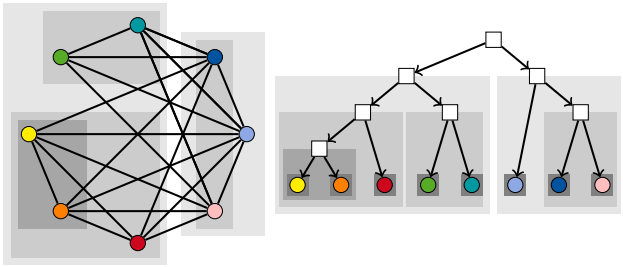

Figure 1: A dense graph of rank width 2 (left) and a hierarchical "rank decomposition" of this graph of width 2 (right). The rank width is low because the vertex cuts the decomposition on the right induces are very regular.

In this paper we study the graph isomorphism problem and the closely related graph canonisation problem as well as logical definability and descriptive complexity on graph classes of bounded rank width. At the technical core of our work is a beautiful connection (from [2]) between the Weisfeiler-Leman algorithm, a generic combinatorial graph isomorphism algorithm, and an Ehrenfeucht-Fraïssé style game that is usually used to prove lower bounds in logic.

While a polynomial time isomorphism algorithm for graph classes of bounded rank width was known [11] prior to this work, the running time of that algorithm is $n^{f(k)}$, where n is the number of vertices and k the rank width of the input graph, and f is a *non-elementary* function. Moreover, the algorithm is extremely complicated, using both advanced techniques from structural graph theory [12] and the group-

The original version of this paper is entitled "Canonisation and Definability for Graphs of Bounded Rank Width" and was published in the proceedings of the Thirty-Fourth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2019).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 2008 ACM 0001-0782/08/0X00 ...$5.00.

theoretic graph isomorphism machinery [19].

Our first contribution is a simple isomorphism test for graphs of rank width at most k running in time $n^{O(k)}$. Indeed, the algorithm we use is a generic combinatorial isomorphism test known as the Weisfeiler-Leman algorithm [23, 1, 2]. The *ℓ-dimensional Weisfeiler Leman* algorithm (ℓ-WL) iteratively colors ℓ-tuples of vertices of the two input graphs and then compares the resulting color patterns. If they differ, we know that the two input graphs are non-isomorphic. If two graphs have the same color pattern, in general they may still be non-isomorphic [2]. Thus ℓ-WL is not a *complete* isomorphism test for all graphs. However, we prove that it is for graphs of bounded rank width. We say that ℓ-WL *identifies* a graph G if it distinguishes G from every graph H not isomorphic to G.

THEOREM 1. *The $(3k+4)$-dimensional Weisfeiler-Leman algorithm identifies every graph of rank width at most k.*

Combining this theorem with a result due to Immerman and Lander on the running time of the WL algorithm, we obtain the following.

COROLLARY 2. *Isomorphism of graphs of rank width k can be decided in time $O(n^{3k+5}\log n)$.*

Another way of stating Theorem 1 is that the *Weisfeiler-Leman (WL) dimension* [8] of graphs of rank width k is at most $3k+4$. It is known that many natural graph classes have bounded WL dimension, among them all classes of graphs excluding some fixed graph as a minor [8]. But most of these classes consist of sparse graphs with an edge number linear in the number of vertices. Our result adds a rich family of classes that include dense graphs to the picture.

In many applications of graph isomorphism, it is actually necessary to compute a canonical representation of a graph, a so-called *canonical form*, rather than just decide if two graphs are isomorphic. A *canonical form* for a graph class $\mathcal{C}$ is a function $\kappa\colon \mathcal{C} \to \mathcal{C}$ such that

1. $\kappa(G) \cong G$ for all $G \in \mathcal{C}$, and
2. $\kappa(G) = \kappa(H)$ for all isomorphic graphs $G, H \in \mathcal{C}$.

Note that the isomorphism problem for a class $\mathcal{C}$ easily reduces to computing a canonical form for $\mathcal{C}$ (i.e., given a graph $G \in \mathcal{C}$, compute $\kappa(G)$). A reduction in the other direction is not known. However, it is known that if a class of graphs has WL dimension at most ℓ then there is an algorithm computing a canonical form for this class running in time $O(n^{\ell+3}\log n)$. Hence, as another corollary to Theorem 1, we obtain the first polynomial time canonisation algorithm for graphs of bounded rank width.

COROLLARY 3. *There is an algorithm computing a canonical form for the class of graphs of rank width at most k in time $O(n^{3k+7}\log n)$.*

The second part of our paper is concerned with descriptive complexity theory, which aims to connect computational complexity and descriptive (or logical) complexity. The central open question of the field is whether there is a logic that *captures polynomial time* [3, 13]. We will defer a more detailed discussion of this question and its background to Section 4 and at this point only state our main result.

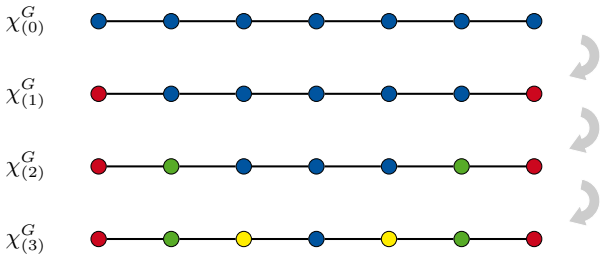

Figure 2: The iterations of the Color Refinement algorithm on a path of length 6.

THEOREM 4. *Fixed-point logic with counting* FP+C *captures polynomial time on every class of graphs of bounded rank width.*

On a very high-level, the connection between the two theorems is that to prove Theorem 4 we mimic the proof of Theorem 1 within the realms of the logic FP+C.

The rest of this article is organized as follows. Section 2 is devoted to a thorough introduction to the Weisfeiler-Leman algorithm. In Section 3 we formally introduce rank width and informally sketch the proof of Theorem 1. Finally, Section 4 is devoted to descriptive complexity theory and Theorem 4.

2. WEISFEILER-LEMAN ALGORITHM

2.1 The Color Refinement Algorithm

One of the simplest combinatorial procedures to tackle the Graph Isomorphism Problem is the Color Refinement algorithm. In a nutshell, the basic idea is to label vertices of the graph with their iterated degree sequence. More precisely, an initially uniform coloring is repeatedly refined by counting, for each color, the number of neighbors of that color.

Let G be a graph. Let $\chi_1, \chi_2\colon V(G) \to C$ be colorings of vertices where C is some finite set of colors. The coloring χ_1 *refines* χ_2, denoted $\chi_1 \preceq \chi_2$, if $\chi_1(v) = \chi_1(w)$ implies $\chi_2(v) = \chi_2(w)$ for all $v, w \in V(G)$. The colorings χ_1 and χ_2 are *equivalent*, denoted $\chi_1 \equiv \chi_2$, if $\chi_1 \preceq \chi_2$ and $\chi_2 \preceq \chi_1$.

We give a formal description of the Color Refinement algorithm. Initially, all vertices are assigned the same color. Formally, we define $\chi^G_{(0)}(v) \coloneqq 0$ for all $v \in V(G)$. Then the coloring is iteratively refined as follows. For $i > 0$ we define $\chi^G_{(i)}(v) \coloneqq (\chi^G_{(i-1)}(v); \mathcal{M}_i(v))$ where

$$\mathcal{M}_i(v) \coloneqq \{\{\chi^G_{(i-1)}(w) \mid w \in N_G(v)\}\}$$

is the multiset of colors for the neighbors computed in the previous round.

In each round, the algorithm computes a coloring that is finer than the one computed in the previous round, i.e., $\chi^G_{(i)} \preceq \chi^G_{(i-1)}$. At some point this procedure stabilizes meaning the coloring does not become strictly finer anymore. In other words, there is some minimal $i^* \in \mathbb{N}$ such that $\chi^G_{(i^*)} \equiv \chi^G_{(i^*+1)}$. We denote the corresponding coloring as the *stable* coloring and define $\chi^G \coloneqq \chi^G_{(i^*)}$. As an example, the sequence of colorings for a path of length 6 is depicted in Figure 2.

The Color Refinement algorithm takes as input a graph G and outputs (a coloring that is equivalent to) χ^G. It can

be implemented in almost linear time, i.e., in time $O((n+m)\log n)$ where n denotes the number of vertices and m the number of edges of G.

One of the most common applications of the Color Refinement algorithm is to serve as a heuristic for GI. Since the coloring computed by the Color Refinement algorithm is defined in an isomorphism-invariant manner, every isomorphism $\varphi\colon G \cong H$ between two graphs G and H satisfies that $\chi^G(v) = \chi^H(\varphi(v))$ for all $v \in V(G)$. The Color Refinement algorithm *distinguishes* G and H if there exists a color c such that

$$\left|\left(\chi^G\right)^{-1}(c)\right| \neq \left|\left(\chi^H\right)^{-1}(c)\right|.$$

In order to determine algorithmically whether the Color Refinement algorithm *distinguishes* G and H one typically runs the Color Refinement algorithm on the disjoint union of G and H and determines whether there is some color c that appears a different number of times in the two graphs.

If the Color Refinement algorithm does not distinguish G and H we write $G \simeq H$. Note that

$$G \cong H \quad \Rightarrow \quad G \simeq H$$

for all graphs G and H. Unfortunately, the backward direction of the implication does not hold. For example, the Color Refinement algorithm does not distinguish the disjoint union of two triangles from a cycle of length six although the two graphs are non-isomorphic.

Still, despite its simplicity, the Color Refinement algorithm serves as a complete isomorphism test for a large number of graphs. We say the Color Refinement algorithm *identifies* a graph G if it distinguishes G from every other non-isomorphic graph H. For example, it is known that the Color Refinement algorithm identifies random graphs asymptotically almost surely.

In combination with its efficiency, this makes the Color Refinement algorithm a popular subroutine in the context of GI which is used in many practical and theoretical algorithms.

2.2 The k-dimensional Algorithm

Although the Color Refinement algorithm often serves as a complete isomorphism test, it is still quite restricted in its power. For example, given any two d-regular graphs G and H, the Color Refinement algorithm does not distinguish G and H. The *Weisfeiler-Leman algorithm* provides an extension to the Color Refinement algorithm which, instead of only coloring single vertices, colors k-tuples of vertices for some fixed number k. This gives the algorithm additional expressive power for increasing values of k, but comes at the price of higher computational complexity.

Let G be a graph and let $k \in \mathbb{N}$. The *k-dimensional Weisfeiler-Leman algorithm* (k-WL) is a procedure that, given a graph G, first computes an isomorphism-invariant initial coloring of the k-tuples of vertices and then iteratively refines this coloring in an isomorphism-invariant way. The 1-dimensional Weisfeiler-Leman algorithm corresponds exactly to the Color Refinement algorithm described above.

For the description of the algorithm for higher dimensions fix $k \geq 2$ and let $G = (V, E)$ be a graph. The initial coloring $\chi^{G,k}_{(0)}$ computed by k-WL determines for each $\bar{v} \in V^k$ the isomorphism-type of the underlying ordered induced subgraph. More precisely, $\chi^{G,k}_{(0)}(v_1,\dots,v_k) = \chi^{G,k}_{(0)}(w_1,\dots,w_k)$ if for all $i,j \in [k]$ it holds $v_i = v_j$ if and only if $w_i = w_j$, and $v_iv_j \in E$ if and only if $w_iw_j \in E$. The initial coloring is refined by iteratively computing colorings $\chi^{G,k}_{(i)}$ for $i > 0$. For $\bar{v} = (v_1,\dots,v_k)$ and $w \in V$ let $\bar{v}[i/w] := (v_1,\dots,v_{i-1},w,v_{i+1},\dots,v_k)$ be the tuple obtained from $\bar{v}$ by replacing the i-th entry with vertex w. Now we define $\chi^{G,k}_{(i)}(\bar{v}) := (\chi^{G,k}_{(i-1)}(\bar{v}); \mathcal{M}_i(v))$ where

$$\mathcal{M}_i(v) := \left\{\!\!\left\{ \left(\chi^{G,k}_{(i-1)}(\bar{v}[1/w]),\dots,\chi^{G,k}_{(i-1)}(\bar{v}[k/w])\right) \,\middle|\, w \in V \right\}\!\!\right\}.$$

From the definition of the colorings it is immediately clear that $\chi^{G,k}_{(i)} \preceq \chi^{G,k}_{(i-1)}$. Now let $i^* \in \mathbb{N}$ be the minimal number such that $\chi^{G,k}_{(i^*)} \equiv \chi^{G,k}_{(i^*+1)}$. For this i^*, the coloring $\chi^{G,k}_{(i^*)}$ is called the *stable* coloring of G with respect to k-WL and is denoted by $\chi^{G,k}$.

As an example, the coloring $\chi^{G,2}$ where G is a cycle of length 9 is depicted in Figure 3.

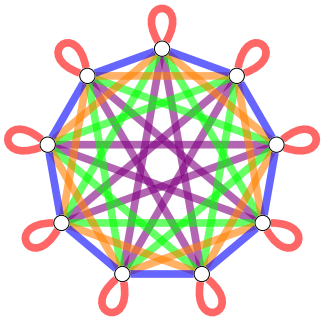

Figure 3: The stable coloring of 2-WL on a cycle of length 9. In this example, $\chi^{G,2}(v,w) = \chi^{G,2}(w,v)$ for all $v,w \in V(G)$ and thus, all colors are represented by undirected edges.

The *k-dimensional Weisfeiler-Leman algorithm* takes as input a graph G and computes (a coloring that is equivalent to) $\chi^{G,k}$.

THEOREM 5 ([17]). *Let G be a graph. Then an isomorphism-invariant coloring that is equivalent to $\chi^{G,k}$ can be computed in time $O(n^{k+1}\log n)$.*

We extend the notations introduced for the Color Refinement algorithm. The k-dimensional Weisfeiler-Leman algorithm *distinguishes* two graphs G and H if there is a color c such that $\left|\left(\chi^{G,k}\right)^{-1}(c)\right| \neq \left|\left(\chi^{H,k}\right)^{-1}(c)\right|$. In this case it follows that G and H are non-isomorphic. Two graphs G and H are *equivalent* with respect to k-WL, denoted $G \simeq_k H$, if they are not distinguished by k-WL. A graph G is *identified* by k-WL if $G \simeq_k H$ if and only if $G \cong H$ for all graphs H. The *Weisfeiler-Leman dimension* of a graph G is the smallest number $k \in \mathbb{N}$ such that k-WL identifies G.

2.3 A Pebble Game

Providing upper bounds on the Weisfeiler-Leman dimension of certain graphs often turns out to be rather complicated when arguing directly with the definition of the Weisfeiler-Leman algorithm. Indeed, it is often more convenient to exploit other characterizations of the power of the Weisfeiler-Leman algorithm for this task. In particular,

the following pebble game that is known to capture the same information as the Weisfeiler-Leman algorithm.

Let $k \in \mathbb{N}$. For graphs G, H on the same number of vertices we define the *bijective k-pebble game* $\mathrm{BP}_k(G,H)$ as follows:

- The game has two players called Spoiler and Duplicator.
- The game proceeds in rounds. Each round is associated with a pair of positions $(\bar{v}, \bar{w})$ with $\bar{v} \in V(G)^\ell$ and $\bar{w} \in V(H)^\ell$ where $0 \leq \ell \leq k$.
- The initial position of the game is $((),())$ (the pair of empty tuples).
- Each round consists of the following steps. Suppose the game is in position $(\bar{v}, \bar{w}) = ((v_1, \dots, v_\ell), (w_1, \dots, w_\ell))$. First, Spoiler chooses whether to remove a pair of pebbles or to play a new pair of pebbles. The first option is only possible if $\ell > 0$ and the latter option is only possible if $\ell < k$.

  If Spoiler wishes to remove a pair of pebbles he picks some $i \in [\ell]$ and the game moves to position $(\bar{v} \setminus i, \bar{w} \setminus i)$ where $\bar{v} \setminus i \coloneqq (v_1, \dots, v_{i-1}, v_{i+1}, \dots, v_\ell)$ (and $\bar{w} \setminus i$ is defined in the same way).

  Otherwise a new pair of pebbles is played and the following steps are performed.

  (D) Duplicator picks a bijection $f \colon V(G) \to V(H)$.

  (S) Spoiler chooses $v \in V(G)$ and sets $w \coloneqq f(v)$.

  The new position is $((v_1, \dots, v_\ell, v), (w_1, \dots, w_\ell, w))$.

  Spoiler wins the play if for the current position $(\bar{v}, \bar{w}) = ((v_1, \dots, v_\ell), (w_1, \dots, w_\ell))$ the induced graphs are not isomorphic. More precisely, Spoiler wins if there are $i, j \in [\ell]$ such that $v_i = v_j \not\Leftrightarrow w_i = w_j$ or $v_i v_j \in E(G) \not\Leftrightarrow w_i w_j \in E(H)$. If the play never ends Duplicator wins.

We say that Spoiler (resp. Duplicator) *wins the bijective k-pebble game* $\mathrm{BP}_k(G,H)$ if Spoiler (resp. Duplicator) has a winning strategy for the game.

Moreover, for positions $(\bar{v}, \bar{w}) \in V(G)^\ell \times V(H)^\ell$, $\ell \leq k$, Spoiler (resp. Duplicator) *wins the game* $\mathrm{BP}_k(G,H)$ *from position* $(\bar{v}, \bar{w})$ if Spoiler (resp. Duplicator) has a winning strategy in the game $\mathrm{BP}_k(G,H)$ started from initial position $(\bar{v}, \bar{w})$.

The next theorem connects the bijective k-pebble game to the Weisfeiler-Leman algorithm.

THEOREM 6 ([2]). *Let G, H be two graphs and let $\bar{v} \in V(G)^k$ and $\bar{w} \in V(H)^k$. Then $\chi^{G,k}(\bar{v}) = \chi^{H,k}(\bar{w})$ if and only if Duplicator wins the pebble game $\mathrm{BP}_{k+1}(G,H)$ from the position $(\bar{v}, \bar{w})$.*

COROLLARY 7. *Let G, H be two graphs. Then $G \simeq_k H$ if and only if Duplicator wins the pebble game $\mathrm{BP}_{k+1}(G,H)$.*

3. RANK WIDTH

Rank width is a graph invariant that was first introduced by Oum and Seymour [21] and which measures the width of a certain style of hierarchical decomposition of graphs. Intuitively, the aim is to repeatedly split the vertex set of the graph along cuts of low complexity in a hierarchical fashion. For rank width, the complexity of a cut is measured in terms of the rank of the matrix capturing the adjacencies between the two sides of the cut over the 2-element field $\mathbb{F}_2$.

Let G be a graph. For $X, Y \subseteq V(G)$ we define $M(X,Y) \in \mathbb{F}_2^{X \times Y}$ where $(M(X,Y))_{x,y} = 1$ if and only if $xy \in E(G)$. Furthermore $\rho_G(X) = \mathrm{rk}_2(M(X, \overline{X}))$ where $\overline{X} = V(G) \setminus X$ and $\mathrm{rk}_2(A)$ denotes the $\mathbb{F}_2$-rank of a matrix A.

A *rank decomposition* of G is a tuple (T, γ) consisting of a binary directed tree T and a mapping $\gamma \colon V(T) \to 2^{V(G)}$ such that

(R.1) $\gamma(r) = V(G)$ where r is the root of T,

(R.2) $\gamma(t) = \gamma(t_1) \cup \gamma(t_2)$ and $\gamma(t_1) \cap \gamma(t_2) = \emptyset$ for all internal nodes $t \in V(T)$ with children t_1 and t_2, and

(R.3) $|\gamma(t)| = 1$ for all $t \in L(T)$, where $L(T)$ denotes the set of leaves of the tree T.

Note that, instead of giving γ, we can equivalently specify a bijection $f \colon L(T) \to V(G)$ (this completely specifies γ by Condition (R.2)). The *width* of a rank decomposition (T, γ) is

$$\mathrm{wd}(T, \gamma) \coloneqq \max\{\rho_G(\gamma(t)) \mid t \in V(T)\}.$$

The rank width of a graph G is

$$\mathrm{rw}(G) \coloneqq \min\{\mathrm{wd}(T, \gamma) \mid (T, \gamma) \text{ is a rank decomp. of } G\}.$$

Examples of graphs G and rank decompositions of G are provided in Figure 1 and (more detailed) in Figure 4.

Clique width [5] is another measure aiming to describe the structural complexity of a graph. It is based on a natural graph grammar.

For $k \in \mathbb{N}$ a *k-graph* is a pair (G, lab) where G is a graph and $\mathrm{lab} \colon V(G) \to [k]$ is a labeling of vertices. We define the following four operations for k-graphs:

1. for $i \in [k]$ let $\bullet_i$ denote an isolated vertex with label i,
2. for $i, j \in [k]$ with $i \neq j$ we define $\eta_{i,j}(G, \mathrm{lab}) \coloneqq (G', \mathrm{lab})$ where $V(G) \coloneqq V(G')$ and
$$E(G') \coloneqq E(G) \cup \{vw \mid \mathrm{lab}(v) = i \wedge \mathrm{lab}(w) = j\},$$
3. for $i, j \in [k]$ we define $\rho_{i \to j}(G, \mathrm{lab}) \coloneqq (G, \mathrm{lab}')$ where
$$\mathrm{lab}'(v) \coloneqq \begin{cases} j & \text{if } \mathrm{lab}(v) = i \\ \mathrm{lab}(v) & \text{otherwise} \end{cases},$$
4. for two k-graphs (G, lab) and (G', lab') we define
$$(G, \mathrm{lab}) \oplus (G', \mathrm{lab}')$$
to be the disjoint union of the two k-graphs.

A *k-expression* t is a well-formed expression in these symbols and defines a k-graph (G, lab). In this case t is a k-expression for G. The clique width of a graph G, denoted by $\mathrm{cw}(G)$, is the minimum $k \in \mathbb{N}$ such that there is a k-expression for G.

Example 1. The expression

$$\eta_{1,2}\Big(\rho_{2\to1}\Big(\eta_{1,2}(\bullet_1 \oplus \bullet_2) \oplus \eta_{1,2}\big(\bullet_2 \oplus \rho_{2\to1}(\eta_{1,2}(\bullet_1 \oplus \bullet_2))\big)\Big) \oplus \rho_{1\to2}\big(\eta_{1,2}((\bullet_1 \oplus \bullet_1) \oplus \bullet_2)\big)\Big)$$

is a 2-expression for the graph in Figure 1. The colors of the "constants" $\bullet_i$ indicate which node in the figure they correspond to.

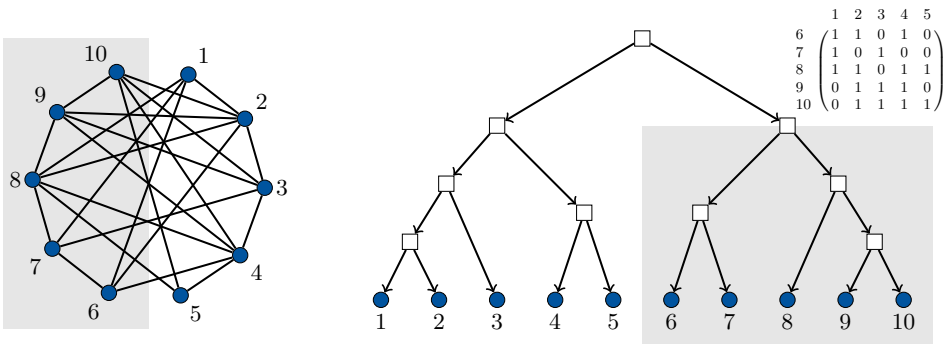

Figure 4: A graph G on the left and a potential rank decomposition of G of width 3 on the right. As an example, the matrix $A_G[\{6,7,8,9,10\},\{1,2,3,4,5\}]$ is provided. Note that $\rho_G(\{6,7,8,9,10\}) = 3$.

Comparing clique width and rank width, each parameter is bounded in terms of the other [21]. More precisely, for every graph G, it holds that

$$\mathrm{rw}(G) \leq \mathrm{cw}(G) \leq 2^{\mathrm{rw}(G)+1} - 1. \tag{1}$$

Also, the rank width of a graph G is bounded by the tree width of G, denoted $\mathrm{tw}(G)$, by $\mathrm{rw}(G) \leq \mathrm{tw}(G) + 1$. Note that the tree width of a graph can not be bounded in terms of its rank width. For example, the complete graph on n vertices K_n has rank width $\mathrm{rw}(K_n) = 1$ and tree width $\mathrm{tw}(K_n) = n - 1$.

3.1 The WL Dimension of Graphs of Bounded Rank Width

The first main result of this work bounds the Weisfeiler-Leman dimension of graphs of rank width at most k by a linear function in k.

THEOREM 8. *The $(3k+4)$-dimensional Weisfeiler-Leman algorithm identifies every graph of rank width at most k.*

Note that, by Equation (1), the same result as well as all of its consequences also hold for graphs of clique width at most k.

For the remainder of this section we provide a high-level sketch of the proof of Theorem 8. We need to argue that, for every two non-isomorphic graphs G and H such that $\mathrm{rw}(G) \leq k$, the $(3k+4)$-dimensional Weisfeiler-Leman algorithm distinguishes between G and H. Exploiting the characterization of k-WL in terms of pebble games (see Corollary 7), it actually suffices to show that Spoiler wins the bijective $(3k+5)$-pebble game $\mathrm{BP}_{3k+5}(G,H)$.

To describe Spoiler's strategy let us fix the input graphs G and H as well as some rank decomposition (T,γ) of G of width at most k. The basic idea for Spoiler is to play along the rank decomposition of G in a downward fashion in order to confine the "difference" between the two input graphs to smaller and smaller regions of the graphs. More precisely, for his strategy, Spoiler maintains sets $X_G \subseteq V(G)$ and $X_H \subseteq V(H)$ and vertices $(a_G^1,\dots,a_G^p,b_G^1,\dots,b_G^p) \in X_G^{2p}$ and $(a_H^1,\dots,a_H^p,b_H^1,\dots,b_H^p) \in X_H^{2p}$ where $p \leq k$ such that there is no isomorphism $\varphi\colon G[X_G] \to H[X_H]$ with $\varphi(a_G^i) = a_H^i$ and $\varphi(b_G^i) = b_H^i$ (where $G[X_G]$ and $H[X_H]$ denote the induced subgraphs on vertex sets X_G and X_H). By gradually decreasing the size of the set X_G Spoiler eventually wins the game since, as soon as $|X_G| \leq 2k+1$, the entire set X_G can be pebbled.

During the play, the sets X_G essentially correspond to the sets $\gamma(t)$ where $t \in V(T)$ is a node of the rank decomposition of G where, as the play progresses, Spoiler moves down the rank decomposition to enforce a decrease in the size of the set X_G. Initially, $X_G := V(G)$, $X_H := V(H)$, $p = 0$, and t is the root node of the rank decomposition of G.

In order to force a descend in the rank decomposition, Spoiler's goal is to move to one of the two children t_1, t_2 of t in the rank decomposition (T,γ). To maintain the invariant described above, it is necessary to "split" $\gamma(t_1)$ from $\gamma(t_2)$ in the graph G in such a way that there are no dependencies left between the two sets. This enables Spoiler to confine the "difference" between the two input graphs to one of the two sides. To achieve the "split" we introduce the notion of a *split pair*.

Let G be a graph and $X \subseteq V(G)$. For $v,w \in X$ we define $v \approx_X w$ if $N(v) \cap \overline{X} = N(w) \cap \overline{X}$. For $v \in X$ we define the vector $\mathrm{vec}_X(v) = (a_{v,w})_{w\in\overline{X}} \in \mathbb{F}_2^{\overline{X}}$ where $a_{v,w} = 1$ if and only if $vw \in E(G)$. Note that $v \approx_X w$ if and only if $\mathrm{vec}_X(v) = \mathrm{vec}_X(w)$. Moreover, for $A \subseteq X$ we define $\mathrm{vec}_X(A) = \{\mathrm{vec}_X(v) \mid v \in A\}$.

For any set of vectors $S \subseteq \mathbb{F}_2^n$ we denote by $\langle S\rangle$ the linear space spanned by S. A set $B \subseteq \mathbb{F}_2^n$ is a *linear basis for* $\langle S\rangle$ if B is linearly independent and $\langle B\rangle = \langle S\rangle$.

Definiton 1. Let G be a graph and $X \subseteq V(G)$. A pair (A,B) is a *split pair for* X if

1. $A \subseteq X$ and $B \subseteq \overline{X}$,
2. $\mathrm{vec}_X(A)$ forms a linear basis for $\langle \mathrm{vec}_X(X)\rangle$, and
3. $\mathrm{vec}_{\overline{X}}(B)$ forms a linear basis for $\langle \mathrm{vec}_{\overline{X}}(\overline{X})\rangle$.

Note that $|A| = \rho_G(X)$ and $|B| = \rho_G(X)$. Also observe that if (A,B) is a split pair for X then (B,A) is a split pair for $\overline{X}$. As a special case the pair $(\emptyset,\emptyset)$ is defined to be a split pair for $X = V(G)$. An *ordered split pair for* X is a pair $(\bar{a},\bar{b}) = ((a_1,\dots,a_p),(b_1,\dots,b_p))$ such that $(\{a_1,\dots,a_p\},\{b_1,\dots,b_p\})$ is a split pair for X.

Now the central claim for split pairs is that pebbling an ordered split pair $(\bar{a},\bar{b})$ for X renders X to be "independent" from its complement. In order to visualize this independence, we make use of the Color Refinement algorithm. Towards this end, we define $\chi^{(\bar{a},\bar{b})} := \chi^{G,(\bar{a},\bar{b})}$ to be the coloring computed by the Color Refinement algorithm on the

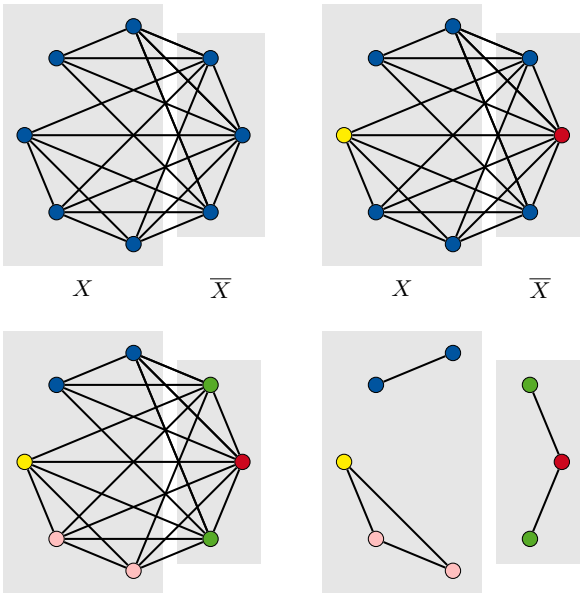

Figure 5: In order to split X from its complement in the graph G (top left) we individualize a split pair (top right), compute the coloring $\chi^{(\bar{a},\bar{b})}$ using the Color Refinement algorithm (bottom left), and apply a suitable flip function to the graph (bottom right).

graph G where all vertices $a_1, \dots, a_p, b_1, \dots, b_p$ are *individualized* initially (i.e., each of the listed vertices is assigned a fresh color that is distinct from the colors of all other vertices). Moreover, we consider the notion of a *flip function* which allows us to flip edges between certain color classes of the coloring $\chi^{(\bar{a},\bar{b})}$.

Definiton 2. Let $G = (V, E)$ be a graph and $\chi\colon V \to C$ a coloring of the vertices. A *flip function for G* is a mapping $f\colon C \times C \to \{0, 1\}$ such that $f(c, c') = f(c', c)$ for all $c, c' \in C$.

For a flip function f we define the flipped graph $G^f = (V, E^f)$ where

$$E^f = \{vw \mid vw \in E \wedge f(\chi(v), \chi(w)) = 0\} \cup \{vw \mid v \neq w \wedge vw \notin E \wedge f(\chi(v), \chi(w)) = 1\}.$$

The following lemma gives the key tool to split X from its complement. For a graph G and a flip function f let $\mathrm{Comp}(G, f) \subseteq 2^{V(G)}$ be the set of vertex sets of the connected components of G^f. Observe that $\mathrm{Comp}(G, f)$ forms a partition of the vertex set of G.

Lemma 9. *Let $G = (V, E)$ be a graph and $X \subseteq V$. Also let $(\bar{a}, \bar{b})$ be an ordered split pair for X.*

Then there is a flip function f for the graph G with coloring $\chi^{(\bar{a},\bar{b})}$ such that for every $C \in \mathrm{Comp}(G, f)$ it holds that $C \subseteq X$ or $C \subseteq \overline{X}$.

This process is also visualized in Figure 5 taking as input the example from Figure 1.

Since applying a flip function to both input graphs neither changes the isomorphism problem nor the outcome of the bijective pebble game Spoiler can simply pretend he is playing on G^f and H^f instead of G and H. For G^f and H^f Spoiler can now restrict the game to a pair of connected components of the two graphs $X_G \subseteq V(G)$ and $X_H \subseteq V(H)$, marking both components by placing a single pebble on them.

Now, in order to force a descend step in the rank decomposition, Spoiler pebbles split pairs for the sets $\gamma(t_1)$ and $\gamma(t_2)$ in order to render these sets to be independent from the rest of the graph. This allows Spoiler to track the difference between both input graphs to one of the two sets.

This almost completes the descend step. Indeed, the only remaining task for Spoiler is to remove all pebbles from previous steps to ensure that the number of required pebbles does not exceed $3k + 5$. In particular, the pebbles placed on the vertices of a split pair for $\gamma(t)$ need to be removed.

However, it is not possible to simply remove these pebbles since, by the invariant described above, the sets X_G and X_H are only assured to be non-isomorphic as long as $(a_G^1, \dots, a_G^p, b_G^1, \dots, b_G^p)$ as well as $(a_H^1, \dots, a_H^p, b_H^1, \dots, b_H^p)$ are pebbled (these vertices need to be temporarily fixed to allow for the applications of Lemma 9). In order to circumvent this problem we introduce the notion of *nice* triples of split pairs.

Definiton 3. Let G be a graph and $X, X_1, X_2 \subseteq V(G)$ such that $X = X_1 \cup X_2$ and $X_1 \cap X_2 = \emptyset$. Let (A, B) be a split pair of X and let (A_i, B_i) be split pairs for X_i, $i \in \{1, 2\}$. We say that (A_i, B_i), $i \in \{1, 2\}$, are *nice (with respect to (A, B))* if

1. $A \cap X_i \subseteq A_i$, and
2. $B_{3-i} \cap X_i \subseteq A_i$

for both $i \in \{1, 2\}$.

Naturally, a triple of ordered split pairs is nice if the underlying unordered triple of split pairs is nice.

Intuitively speaking, nice triples of split pairs enforce a significant overlap of all considered split pairs when performing the descend step. This enables Spoiler to remove the pebbles from the previous descend step without unpebbling any vertex in X_G or X_H. This following lemma guarantees the existence of nice triples of split pairs.

Lemma 10. *Let G be a graph and $X, X_1, X_2 \subseteq V(G)$ such that $X = X_1 \cup X_2$ and $X_1 \cap X_2 = \emptyset$. Let (A, B) be a split pair of X. Then there are nice split pairs (A_i, B_i) for X_i for both $i \in \{1, 2\}$.*

Hence, in all steps, Spoiler can play a nice triple of split pairs and simply remove any unwanted pebbles following the completion of a descend step while preserving the invariant given above. This completes the description of the strategy.

4. DESCRIPTIVE COMPLEXITY

Descriptive complexity theory aims to characterize computational complexity in terms of logical definability, that is, relate computational resources such as space and time to language resources such as recursion mechanisms or quantifier alternation. The best-known result in the field is Fagin's Theorem [7] stating existential second-order logic captures NP, that is, a property of graphs (or other structures) is decidable in nondeterministic polynomial time if and only if it is definable in existential second-order logic. The best-known open problem, going back to Chandra and Harel's

influential paper [3] on database query languages and popularized by Gurevich [13], asks whether there is a logic that captures PTIME (polynomial time). In the following, we give a very brief introduction to the relevant notions and results form descriptive complexity. For more background, we refer the reader to the existing literature (e.g., [6]).

In descriptive complexity theory, computational problems are typically modeled using finite relational structures, for example, graphs. A decision problem is simply a *property of structures*, that is, a class of structures that is closed under isomorphism. Closure under isomorphism is important, because it means that properties are isomorphism-invariant. For example, "a graph is connected" is a valid property, whereas "the first vertex has degree 3" is not, because a graph has no well-defined "first vertex". An *abstract logic* L is simply a set of sentences together with a satisfaction relation between structures and sentences. An abstract logic L *captures* PTIME if (i) it has a decidable syntax (the set of sentences is a decidable set of strings over a finite alphabet), (ii) it has a PTIME-decidable semantics, in the sense that there is an algorithm that, given a sentence ϕ of L and a structure A decides if A satisfies ϕ in time polynomial in the size of A, and (iii) every PTIME-decidable property $\mathcal{P}$ of structures is definable in L, that is, there is a sentence of L that is satisfied by precisely those structures that have property $\mathcal{P}$. We say L captures PTIME *on a class $\mathcal{C}$ of structures* if condition (iii) is satisfied for all properties $\mathcal{P} \subseteq \mathcal{C}$.

While there are good reasons to define an abstract logic capturing PTIME this way (see [13]), the reason we are interested in a logical characterization of PTIME is the hope that it might give us new insights into the mechanisms of efficient computation. For this, we are not so interested in an abstract logic, but rather in nice logical languages with few clearly understood operators. Logics that have proved to be natural and useful in this regard are extensions of classical first-order logic FO by fixed-point operators allowing it to formalize iterative or recursive definitions. *Inflationary fixed-point logic* FP is a robust extension of FO with a fixed-point operator that allows it to define a relation by iteratively adding tuples to the initially empty relation.

Example 2. The FP-sentence conn defined as

$$\forall x \forall y \ \text{ifp}\Big(Z \leftarrow z \;\Big|\; z = x \vee \exists z'\big(Z(z') \wedge E(z', z)\big)\Big)(y).$$

expresses that a graph is connected. For any two nodes x, y, the ifp$(\cdots)$ operator computes the connected component X of x by adding x and then in each iteration all neighbors of nodes already in X. Then the sentence conn says that for all x, y, y belongs to the connected component X of x.

Often, it is useful to also add rudimentary arithmetic to the logic, in particular, the ability to count. *Inflationary fixed-point logic with counting* FP+C is an extension of FP by the ability to speak about an initial segment of the natural numbers with basic arithmetic and a counting mechanism relating structures and numbers.

Example 3. The following FP+C-sentence defines the class of Eulerian graphs (that is, graphs with a cyclic walk that traverses all edges, which are well-known to be exactly the connected graphs in which all vertices have even degree):

$$\text{eulerian} := \text{conn} \wedge \forall x \ \text{even}\big(\#y \ E(x, y)\big).$$

Here conn is the sentence defined in the previous example, $\#y \ E(x, y)$ defines the number of nodes y that are adjacent to node x, and even(n) is a formula expressing that n is an even number.

So FP and FP+C extend first-order logic, which provides the basic logical machinery, by two fundamental computational mechanisms: iteration and counting. At first sight it may seem that counting the number of elements in a set, for example the set of neighbors of a vertex in a graph, can be simulated by enumerating the elements in an iterative process implemented in FP. But this assumes that the elements of the set can be put in some order, and such an order is not always available in a structure. Logics operate in an isomorphism-invariant way, and there may be no isomorphism-invariant order on our set.

While FP and FP+C fail to capture PTIME—this is easy to prove for FP and surprisingly hard for FP+C [2]—, it has been shown that they can express many natural PTIME-properties and they do capture PTIME restricted to a large variety of classes of structures. As a starting point, Immerman [15] and independently Vardi [22] proved that FP captures PTIME on the class of ordered structures, that is, structures that have one relation which is a linear order of the universe. Note that on ordered structures we can simulate counting using iteration, thus FP and FP+C have the same expressive power. As soon as we leave ordered structures, we need the explicit counting mechanism of FP+C. In a series of papers that culminated in the monograph [8], it was proved that FP+C captures PTIME on all graph classes that exclude a fixed graph as a minor. In particular, this includes the class of planar graphs and all graph classes of bounded tree width. Not too much is known beyond these classes, which all consist of sparse graphs. Indeed, looking at dense graphs, there are only very few examples of classes for which it is known that FP+C captures PTIME, most of which are fairly restricted classes of intersection graphs (e.g., interval graphs, permutation graphs). Our second main result, Theorem 4 stating that FP+C captures PTIME on all graph classes of bounded rank width, broadens the scope of these results into a new direction.

Like previous results of this type, Theorem 4 is proved using the framework of *definable canonisation* [8, 20]. Recall that by the Immerman-Vardi Theorem [16, 22], FP captures polynomial time on the class of all ordered structures. A straightforward way of applying this theorem to a class $\mathcal{C}$ of unordered structures is to define a linear order on this class: if there is a formula ord(x, y) of the logic FP that defines a linear order on a all structures in $\mathcal{C}$, then FP still captures polynomial time on $\mathcal{C}$. Unfortunately, this observation is rarely applicable, because usually it is impossible to define linear orders. For example, it is impossible to define a linear order on a structure that has a non-trivial automorphism.

A more refined idea, known as *definable canonisation*, is to define an *ordered copy* of the input structure. To implement this idea, FP+C is particularly well-suited, because it allows us to speak about an initial segment of the natural numbers that comes with a natural linear order. Technically, definable canonisation is based on syntactical interpretations, which allow it to define a structure within another structure using logical formulas. (In database terminology, a syntactical interpretation would be a view.)

Without going into any details, let us just state that the main technical lemma towards the proof of Theorem 4 states

that for all k the class of all graphs of rank width k admits FP+C-definable canonisation. The idea to prove this lemma is to implement our canonisation procedure for graphs of rank width k based on the $(3k+4)$-dimensional Weisfeiler-Leman algorithm by a formula of the logic FP+C.

For all further details, we refer the reader to the full version of this article [10].

5. CONCLUSIONS

In this paper we considered the isomorphism and canonisation problem for graphs of bounded rank width. The first main result is that the Weisfeiler-Leman dimension of graphs of rank width at most k is at most $3k+4$. This implies that isomorphism testing and computing canonical forms for graphs of rank width at most k can be done in time $n^{O(k)}$.

Naturally, it would be desirable to further improve on the complexity of the two problems. Indeed, an important open question is whether isomorphism testing is also fixed-parameter tractable when parameterized by rank width (i.e., whether there is an algorithm testing isomorphism of graphs of rank width at most k in time $f(k)n^c$ for some computable function f and an absolute constant c).

The second main result states that fixed-point logic with counting captures polynomial time on the class of graphs of rank width at most k.

6. REFERENCES

[1] L. Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In D. Wichs and Y. Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 684–697. ACM, 2016.

[2] J. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identifications. *Combinatorica*, 12(4):389–410, 1992.

[3] A. K. Chandra and D. Harel. Structure and complexity of relational queries. *J. Comput. Syst. Sci.*, 25(1):99–128, 1982.

[4] B. Courcelle, J. A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.

[5] B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

[6] H. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer Verlag, 2nd edition, 1999.

[7] R. Fagin. Generalized first–order spectra and polynomial–time recognizable sets. In R. Karp, editor, *Complexity of Computation, SIAM-AMS Proceedings, Vol. 7*, pages 43–73, 1974.

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

[9] M. Grohe and D. Marx. Structure theorem and isomorphism test for graphs with excluded topological subgraphs. *SIAM J. Comput.*, 44(1):114–159, 2015.

[10] M. Grohe and D. Neuen. Canonisation and definability for graphs of bounded rank width. *CoRR*, abs/1901.10330, 2019.

[11] M. Grohe and P. Schweitzer. Isomorphism testing for graphs of bounded rank width. In V. Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1010–1029. IEEE Computer Society, 2015.

[12] M. Grohe and P. Schweitzer. Computing with tangles. *SIAM J. Discrete Math.*, 30(2):1213–1247, 2016.

[13] Y. Gurevich. Logic and the challenge of computer science. In E. Börger, editor, *Current trends in theoretical computer science*, pages 1–57. Computer Science Press, 1988.

[14] J. E. Hopcroft and R. Tarjan. Efficient planarity testing. *J. ACM*, 21:549–568, 1974.

[15] N. Immerman. Relational queries computable in polynomial time (extended abstract). In *Proceedings of the 14th ACM Symposium on Theory of Computing*, pages 147–152, 1982.

[16] N. Immerman. Languages that capture complexity classes. *SIAM J. Comput.*, 16(4):760–778, 1987.

[17] N. Immerman and E. Lander. Describing graphs: A first-order approach to graph canonization. In A. L. Selman, editor, *Complexity Theory Retrospective: In Honor of Juris Hartmanis on the Occasion of His Sixtieth Birthday, July 5, 1988*, pages 59–81. Springer New York, New York, NY, 1990.

[18] R. Karp. On the complexity of combinatorial problems. *Networks*, 5:45–68, 1975.

[19] E. M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comput. Syst. Sci.*, 25(1):42–65, 1982.

[20] M. Otto. *Bounded variable logics and counting – A study in finite models*, volume 9 of *Lecture Notes in Logic*. Springer, 1997.

[21] S. Oum and P. D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory, Ser. B*, 96(4):514–528, 2006.

[22] M. Y. Vardi. The complexity of relational query languages (extended abstract). In H. R. Lewis, B. B. Simons, W. A. Burkhard, and L. H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 137–146. ACM, 1982.

[23] B. Weisfeiler and A. Leman. The reduction of a graph to canonical form and the algebra 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`.