FPSAC 2013 Paris, France DMTCS proc. **AS**, 2013, 767–778

Generating tuples of integers modulo the action of a permutation group and applications

Nicolas Borie

Univ. Paris Est Marne-La-Vallée, Laboratoire d'Informatique Gaspard Monge, Cité Descartes, France

---

Abstract. Originally motivated by algebraic invariant theory, we present an algorithm to enumerate integer vectors modulo the action of a permutation group. This problem generalizes the generation of unlabeled graph up to an isomorphism. In this paper, we present the full development of a generation engine by describing the related theory, establishing a mathematical and practical complexity, and exposing some benchmarks. We next show two applications to effective invariant theory and effective Galois theory.

Résumé. Initialement motivé par la théorie algébrique des invariants, nous présentons une stratégie algorithmique pour énumérer les vecteurs d'entiers modulo l'action d'un groupe de permutations. Ce problème généralise le problème d'énumération des graphes non étiquetés. Dans cet article, nous développons un moteur complet d'énumération en expliquant la théorie sous-jacente, nous établissons des bornes de complexité pratiques et théoriques et exposons quelques bancs d'essais. Nous détaillons ensuite deux applications théoriques en théorie effective des invariants et en théorie de Galois effective.

Keywords: Generation up to an Isomorphism, Enumerative Combinatorics, Computational Invariant Theory, Effective Galois Theory

---

1 Introduction

Let G be a group of permutations, that is, a subgroup of some symmetric group $\mathfrak{S}_n$. Several problems in effective Galois theory (see [Girstmair(1987), Abdeljaouad(2000)]), computational commutative algebra (see [Faugère and Rahmany(2009), Borie and Thiéry(2011), Borie(2011)]) and generation of unlabeled with repetitions species of structures rely on the following computational building block.

Let $\mathbb{N}$ be the set of non-negative integers. An *integer vector* of length n is an element of $\mathbb{N}^n$. The symmetric group $\mathfrak{S}_n$ acts on positions on integer vectors in $\mathbb{N}^n$: for σ a permutation and $(v_1, \ldots, v_n)$ an integer vector,

$$\sigma.(v_1, \ldots, v_n) := (v_{\sigma^{-1}(1)}, \ldots, v_{\sigma^{-1}(n)}) .$$

This action coincides with the usual action of $\mathfrak{S}_n$ on monomials in the multivariate polynomial ring $\mathbb{K}[\mathbf{x}]$ with $\mathbb{K}$ a field and $\mathbf{x} := x_1, \ldots, x_n$ indeterminates.

Problem 1.1 *Let $G \subset \mathfrak{S}_n$ be a permutation group. Enumerate the integer vectors of length n modulo the action of G.*

Note that there are infinitely many such vectors; in practice one usually wants to enumerate the vectors with a given sum or content.

For example, the Problem 1.1 contains the listing non-negative integer matrices with fixed sum up to the permutations of rows or columns appearing in the theory of multisymmetric functions [Gessel(1987), MacMahon(2004)] and in the more recent investigations of multidiagonal coinvariant [Bergeron(2009), Bergeron et al.(2011)Bergeron, Borie, and Thiéry].

Define the following equivalence relation over elements of $\mathbb{N}^n$: two vectors $\mathbf{u} := (a_1, \dots, a_n)$ and $\mathbf{v} := (b_1, \dots, b_n)$ are equivalent if there exists a permutation $\sigma \in G$ such that

$$\sigma \cdot \mathbf{u} = (a_{\sigma^{-1}(1)}, \dots, a_{\sigma^{-1}(n)}) = (b_1, \dots, b_n) = \mathbf{v}.$$

Problem 1.1 consists in enumerating all $\mathbb{N}^n/G$ equivalence classes.

This problem is not well solved in the literature. Some applications present a greedy strategy searching and deleting all pairs of vectors such that the second part can be obtained from the first part. The most famous sub-problem is the unlabeled graph generation which consists in enumerate tuples over 0 and 1 of length $\binom{n}{2}$ enumerated up to the action of the symmetric groups acting on pair on nodes. This example has a very efficient implementation in Nauty which is able to enumerate all graphs over a small number of nodes.

The algorithms presented in this paper have been implemented, optimized, and intensively tested in `Sage` [Stein et al.(2009)]; most features are integrated in `Sage` since release 4.7 (2011-05-26, ticket #6812, 1303 lines of code including documentation).

2 Orderly generation and tree structure over integer vectors

The orderly strategy consists in setting a total order on objects before quotienting by the equivalence relation. This allows us to define a single representative by orbit. Using the lexicographic order on integer vectors, we will call a vector $\mathbf{v}$ *canonical under the action of* G or just *canonical* if $\mathbf{v}$ is maximum in its orbit under G for the lexicographic order:

$$\mathbf{v} \text{ is canonical } \Leftrightarrow \mathbf{v} = \max_{lex}\{\sigma \cdot \mathbf{v} \mid \sigma \in G\}.$$

Now, the goal being to avoid to test systematically if vectors are canonical, we decided to use a tree structure on the objects in which we will get properties relaying the *canonical* vectors. Any result relating fathers, sons and the property of being canonical in the tree may allow us to skip some canonical test.

2.1 Tree Structure over integer vectors

Let $\mathbf{r}$ be the vector $\mathbf{r} := (0, \dots, 0)$ called *root*, we build a tree with the following function *father*.

Definition 2.1 *Let $\mathbf{a} = (a_1, a_2, \dots, a_n)$ be a tuple of integers of length n which is not the* root*. Let $1 \leqslant i \leqslant n$ be the position of the last non-zero entry of* $\mathbf{a}$*. We define the* father *of* $\mathbf{a}$

$$father(a_1, a_2, \dots, a_i, 0, 0, \dots, 0) := (a_1, a_2, \dots, a_i - 1, 0, 0, \dots, 0)$$

For any integer vector $\mathbf{v} = (a_1, \ldots, a_n)$, we can go back to the generation root $(0, \ldots, 0)$ by $sum(\mathbf{v}) := a_1 + \cdots + a_n$ steps. The corresponding application giving the children of an integer vector is thus:

Definition 2.2 *Let $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ be a tuple of integers of length n. Let $1 \leqslant i \leqslant n$ be the position of the last non-zero entry of $\mathbf{a}$ ($i = 1$ if all entries are null). The set of* children *of $\mathbf{a}$ is obtained as:*

$$\textit{children: } (a_1, a_2, \ldots, a_i, 0, 0, \ldots, 0) \longmapsto \left\{ \begin{array}{c} (a_1, a_2, \ldots, a_i + 1, 0, 0, \ldots, 0) \\ (a_1, a_2, \ldots, a_i, 1, 0, \ldots, 0) \\ (a_1, a_2, \ldots, a_i, 0, 1, \ldots, 0) \\ \cdots \\ (a_1, a_2, \ldots, a_i, 0, 0, \ldots, 1) \end{array} \right\}$$

Proposition 2.3 *For any permutation group $G \subset \mathfrak{S}_n$, for any integer vector $\mathbf{v}$, if $\mathbf{v}$ is not canonical under G, all children of $\mathbf{v}$ are not canonical. Therefore, the* canonicals *form a "prefix tree" in the tree of all integer vectors.*

Sketch of proof: *When a father is not canonical, there exists a permutation such that the permuted vector is greater. Applying the same permutation on the children shows also it cannot be canonical.*

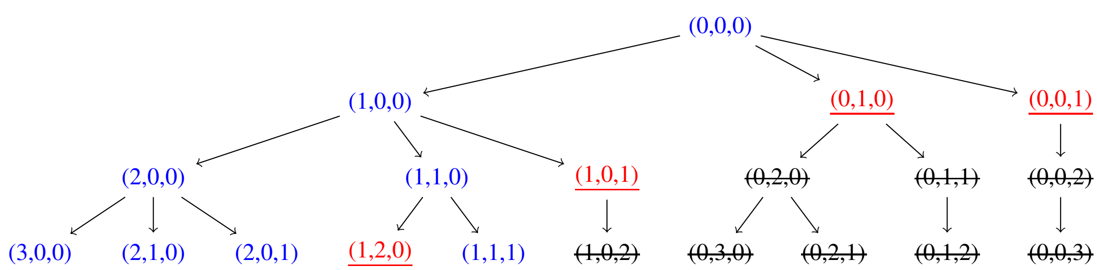


Figure 1: Enumeration tree of integer vectors modulo the action of $G = \langle(1, 2, 3)\rangle \subset \mathfrak{S}_3$, the cyclic group of degree 3.

Figure 1 displays integer vectors of length 3 whose sum is at most 3 and shows the tree relations between them. Choosing the cyclic group of order 3 and using the generation strategy, underlined integer vectors are tested but are recognized to be not *canonical*. Using Proposition 2.3, crossed-out integer vectors are not tested as they cannot be *canonical* as children of non canonical vectors.

Our strategy consists now in making a breath first search over the sub-tree of *canonicals*. This is done lazily using `Python` iterators.

2.2 Testing whether an integer vector is canonical

As we have seen, the fundamental operation for orderly generation is to test whether an integer vector is canonical; it is thus vital to optimize this operation. To this end, we use the work horse of computational group theory for permutation groups: stabilizer chains and strong generating sets.

Following the needs required by applications, we want to test massively if vectors are canonical or not. For this reason, we will use a *strong generating system* of the group G. We can compute this last item in almost linear time [Seress(2003)] using `GAP` [GAP(1997)].

Let n a positive integer and G a permutation group $G \subset \mathfrak{S}_n$. Recall that its *stabilizer chain* is $G_n = \{e\} \subset G_{n-1} \subset \cdots \subset G_1 \subset G_0 = G$, where

$$\forall i, 1 \leqslant i \leqslant n : G_i := \{g \in G | \forall j \leqslant i : g(j) = j\}\,.$$

From this chain, we build a *strong generating system* $T = \{T_1, T_2, \ldots, T_n\}$ where T_i is a transversal of G_{i-1}/G_i. This set of strong generators is particularly adapted to the partial lexicographic order as stabilizers are defined with positions $1, 2, \ldots, n$ from left to right.

Let n and i be two positive integers such that $1 \leqslant i \leqslant n$. For $\mathbf{v} = (v_1, \ldots, v_n)$ and $\mathbf{w} = (w_1, \ldots, w_n)$ two integer vectors of length n, let us define the following binary relations

$$\begin{array}{lcl} \mathbf{v} <_i \mathbf{w} & \iff & (v_1, \ldots, v_i) <_{lex} (w_1, \ldots, w_i) \\ \mathbf{v} \leqslant_i \mathbf{w} & \iff & (v_1, \ldots, v_i) \leqslant_{lex} (w_1, \ldots, w_i) \\ \mathbf{v} =_i \mathbf{w} & \iff & \forall j, 1 \leqslant j \leqslant i : v_j = w_j \end{array}$$

where $<_{lex}$ and $\leqslant_{lex}$ represent regular strict and large lexicographic comparison.

Algorithm 1 is a natural extension of McKay's canonical graph labeling algorithm as it is explained in [Hartke and Radcliffe(2009)].

Algorithm 1 Testing whether an integer vector is canonical

Arguments

- $\mathbf{v}$: An integer vector of length n;
- $sgs(G)$: A strong generating set for G, as a list $\{T_1, \ldots, T_n\}$ of transversals.

```
def is_canonical(v, sgs(G)) :
    todo ← {v}
    for i ∈ {1, 2, ..., n} :
        new_todo ← { }
        for w ∈ todo :
            children ← {g · w | g ∈ T_i}
            for child ∈ children :
                if v <_i child :
                    return False
                else :
                    if v =_i child and child ∉ new_todo :
                        new_todo ← new_todo ∪ {child}
        todo ← new_todo
    return True
```

Algorithm 1 takes advantage of partial lexicographic orders and the *strong generating system* of the group G. It tries to explore only a small part of the orbit of the vector $\mathbf{v}$; the worst case complexity of this step is bounded by the size of the orbit, and not by $|G|$. In this sense, it does take into account the automorphism group of the vector $\mathbf{v}$.

Proposition 2.4 *Let n be a positive integer and G a subgroup of $\mathfrak{S}_n$. Let $\mathbf{v}$ be an integer vector of length n. Algorithm 1 returns $True$ if $\mathbf{v}$ is* canonical *under the action of G and returns $False$ otherwise.*

Sketch of proof: *It is based on the properties of a* strong generating system.

3 Complexity

3.1 Theoretical complexity

3.1.1 Efficiency of the tree structure

Let n be a positive integer and $G \subset \mathfrak{S}_n$ a permutation group. For any non negative integer d, let $C(d)$ (resp. $\overline{C}(d)$) be the number of canonical (resp. non canonical) integer vectors of degree d. Based on the tree structure presented in Section 2.1, let $T(n)$ (resp. $\overline{T}(n)$) the number of tested (resp. non tested) integer vectors.

Proposition 3.1 *Generating all canonical integer vectors up to degree $d \geqslant 0$ using the generation strategy presented in Section 2 presents an* absolute error *bounded by $\overline{C}(d)$. Equivalently, regarding the series, we have*

$$\sum_{i=0}^{d} T(i) - \sum_{i=0}^{d} C(i) \leqslant \overline{C}(d)$$

Sketch of proof: *Using Lemma 2.3, we get this bound noticing two tested but non canonical vectors cannot have a paternity relation.*

This *absolute error* is not very explicit (directly usable), but it can be used to get a *relative error* at the price of a rough approximation.

Corollary 3.2 *Let n and b be two positive integers and $G \subset \mathfrak{S}_n$ a permutation group. Generating all canonical monomials under the action of G up to degree d using the generation strategy presented in Section 2 presents a* relative error *bounded by* $\min\{\frac{n(|G|-1)}{n+d}, n-1\}$.

Sketch of proof: *We use the previous proposition with the fact that any integer vector has at least one child but no more than $n-1$ children (the generation root is the only one having n children).*

The bound is optimal for trivial groups ($\{e\} \subset \mathfrak{S}_n$), and seems to be better as the permutation group is of small cardinality. This relative error becomes better as we go up along the degree and tends to become optimal when the degree goes to infinity.

3.1.2 Complexity of testing if a vector is canonical

We now investigate the complexity of Algorithm 1. We need first to select a reasonable statistic to collect, which will define the complexity of this algorithm.

The explosion appearing in the algorithm is conditioned by the size of the set new_todo. For $\mathbf{v}$ an integer vector and $\{T_1, \ldots, T_n\}$ a *strong generating system* of a permutation G, when i runs over $\{1, 2, \ldots, n\}$ in the main loop, the set new_todo_i contains at step i:

$$new\_todo_i = \{g_1 \cdots g_i \cdot \mathbf{v} | g_1 \cdots g_i \cdot \mathbf{v} =_i \mathbf{v}, \forall j \leqslant i : g_j \in T_j\}$$

The right statistic to record is the size of the union of the new_todo_i for all i such that the algorithm is still running: that corresponds to the part of the orbit explored by the algorithm. This statistic appears to be very difficult to evaluate by a theoretical way. However, collecting it with a computer is a simple task.

3.1.3 Parallelization and memory complexity

Let us note that this generation engine is trivially amenable for parallelism: one can devote the study of each branch to a different processor. Our implementation uses a little framework `SearchForest`, co-developed by the author, for exploration trees and map-reduce operations on them. To get a parallel implementation, it is sufficient to use the drop-in parallel replacement for `SearchForest` under development by Jean-Baptiste Priez and Florent Hivert.

The memory complexity of the generation engine is reasonable, bounded by the size of the answer. Indeed, we keep in the cache only the *Canonical* vectors of degree $d-1$ when we search for those in degree d. In case one wants to only *iterate* through the elements of a given degree d, then this can be achieved with memory complexity $O(nd)$.

3.2 Benchmarks design

To benchmark our implementation, we chose the following problem as test-case.

Problem 3.3 *Let n be a positive integer and $G \subset \mathfrak{S}_n$ a permutation group. Iterate through all the canonical integer vectors v under the staircase (i.e. $v_i \leq n-i$).*

A vector $\mathbf{v}$ of length n is said to be *under the staircase* when it is componentwise smaller than the vector $(n-1, n-2, \dots 1, 0)$.

This problem contains essentially all difficulties that can appear. The family of $n!$ integer vectors under the staircase contains vectors with trivial automorphism group as well as vectors with a lot of symmetries. Applications also require to deal with this problem as the corresponding family of monomials plays a crucial role in algebra.

3.2.1 Benchmarks for transitive permutation groups

We now need a good family of permutation groups, representative of the practical use cases. We chose to use the database of all transitive groups of degree ≤ 30 [Hulpke(2005)] available in `Sage` through the system `GAP` [GAP(1997)].

The benchmarks have been run on an off-the-shelf `2.40 GHz` dual core Mac Book laptop running `Ubuntu 12.4` and `Sage` version 5.3.

3.3 Benchmarks

3.3.1 Tree Structure over integer vectors

This first benchmark investigates the efficiency of the tree structure presented in Section 2.1. As we don't test children of non canonical integer vectors, one wants to take measures of the part of tested non canonical vectors (which corresponds to the useless part of computations). For that, we solve Problem 3.3 for each group of the database and we collect the following information as follows.

Transitive Groups of degree 5				
Database Id.	$\|G\|$	Index in $\mathfrak{S}_n$	*Canonicals*	number of tests
1	5	24	71	81
2	10	12	68	81
3	20	6	46	67
4	60	2	41	67
5	120	1	41	67

This table displays the statistics for transitive groups of degree 5. *Database Id.* is the integer indexing the group, $|G|$ and *Index in* $\mathfrak{S}_n$ are respectively the cardinality and the index of the group G in the symmetric group $\mathfrak{S}_n$. *Canonicals* denotes the number of canonical vectors under the staircase and *number of tests* is the number of times the algorithm testing if an integer vector is canonical is called.

From this information, we set a quantity Err defined as follows:

$$Err := \frac{\text{number of tests} - \text{Canonicals}}{\text{Canonicals}}.$$

The following figure shows Err depending on the index $\frac{n!}{|G|}$. The figure contains 166 crosses, one for each transitive group over at most 10 variables. We use a logarithmic scale on the x axis.

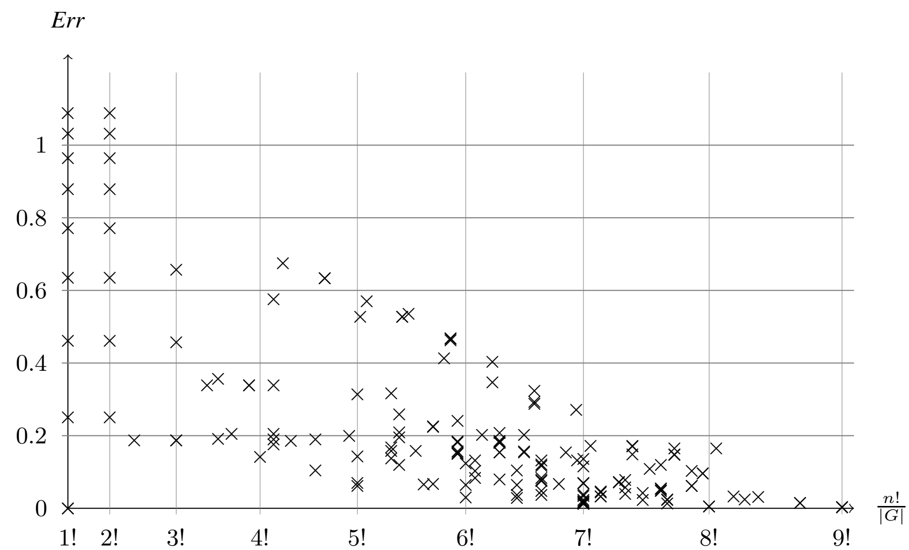


Figure 2: Relative Error between number of tested vectors and number of *canonicals* vectors.

3.3.2 Empirical complexity of testing if a vector is canonical

Algorithm 1 needs to explore a part of the orbit of the tested integer vectors. The following table displays for each transitive group over 5 variables, the number of elements of all orbits of tested vectors solving

Problem 3.3 compared to the total number of integer vectors explored.

Transitive Groups of degree 5				
Database Id.	$\lvert G\rvert$	Index in $\mathfrak{S}_n$	total orbits	total explored
1	5	24	401	351
2	10	12	691	393
3	20	6	1091	365
4	60	2	1891	328
5	120	1	1891	326

Now we define $Ratio$ to be the average size of the orbit needed to be explored to know if an integer vector is canonical:

$$Ratio := \frac{\text{total explored}}{\text{total orbits}}.$$

The following figure plots $Ratio$ in terms of $|G|$ for transitive groups on at most 9 variables.

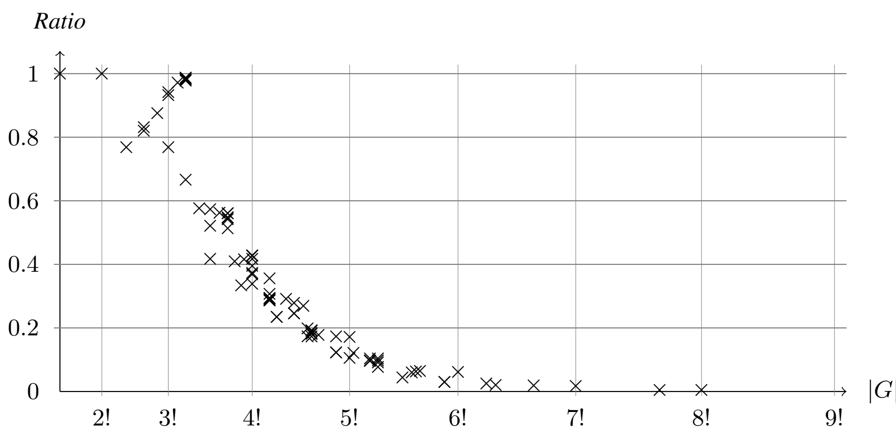


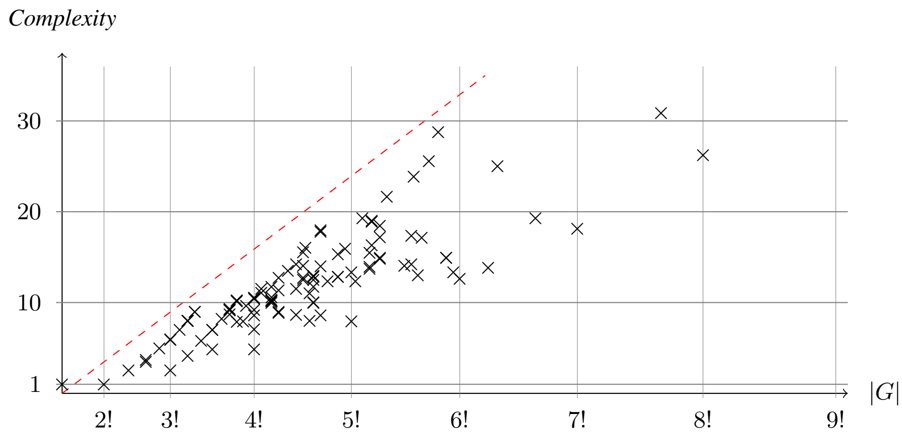
Figure 3: Average, over all integer vectors v under the stair case, of the number of vectors in the orbit of v explored by `is_canonical(v)`.

3.3.3 Overall empirical complexity of the generation engine

We now evaluate the overall complexity by comparing the ratio between the computations and the size of the output. We define the measure *Complexity* as follows:

$$Complexity := \frac{\text{total explored}}{\text{Canonicals}}.$$

The following graph displays *Complexity* in terms of the size of the group $|G|$ for transitive Groups on up to 9 variables (and excluding the alternate and symmetric group of degree 9).



The dashed line has as equation $y = 5ln(|G|)$. Therefore, we get the following empirical overall complexity:

$$\text{Computations} = O(ln(|G|) \times \text{Output size})$$

3.3.4 Tests around the unlabeled graph generation problem

Although the generation engine is not optimized for the unlabeled graph generation problem, we can apply our strategy on it.

Fix n, and consider the set E of pairs of elements of n. The symmetric group $\mathfrak{S}_n$ acts on pairs by $\sigma \cdot (i,j) = (\sigma(i), \sigma(j))$ for $\sigma \in \mathfrak{S}_n$ and $(i,j) \in E$. Let G be the induced group of permutations of E. A labeled graph can be identified with the integer vector with parts in $0, 1$. Then, two graphs are isomorphic if and only if the corresponding vectors are in the same G-orbit.

Now, one needs just to know which are these permutation groups acting on pairs of integers. In the following example, we retrieve the number of graphs on n unlabeled nodes is, for small values of n is given by: 1, 1, 2, 4, 11, 34, 156, 1044, 12346, 274668, 12005168, ...

```
sage: L = [TransitiveGroup(1,1), TransitiveGroup(3,2),
TransitiveGroup(6,6), TransitiveGroup(10,12), TransitiveGroup(15,28),
TransitiveGroup(21,38), TransitiveGroup(28,502)]
sage: [IntegerVectorsModPermutationGroup(G,max_part=1).cardinality() for G in
L]

[2, 4, 11, 34, 156, 1044, 12346]
```

Notice that our generation engine generalizes the graph generation problem in two directions. Removing the option `max_part`, one enumerates multigraphs (graphs with multiple edges between nodes). On the other hand, graphs correspond to special cases of permutation groups. From an algebraic point of view, we saw graphs as monomials whose exponents are 0 or 1, canonical for the action of the symmetric group on pairs of nodes.

4 Computing the invariants ring of a permutation group

Let us explain how the generation engine from Section 2 is plugged into effective invariant theory (see [Derksen and Kemper(2002)] and [King(2007)]).

A well-known application to build an *invariant polynomial* under the action of a permutation group G is the Reynolds operator R. From any polynomial P in n variables $\mathbf{x} := x_1, x_2, \ldots, x_n$, the invariant is

$$R(P) := \frac{1}{|G|} \sum_{\sigma \in G} \sigma \cdot P,$$

where $\sigma \cdot P$ is the polynomial built from P for which σ has permuted by position the tuple of variables $(x_1, x_2, \ldots, x_n)$. Formally, for any $\sigma \in G$

$$(\sigma \cdot P)(x_1, x_2, ..., x_n) := P(x_{\sigma^{-1}(1)}, x_{\sigma^{-1}(2)}, \ldots, x_{\sigma^{-1}(n)}).$$

For large groups, the Reynolds operator is not very convenient to build invariant polynomials. If P is a monomial $\mathbf{x}^{\mathbf{a}} := x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}$ where $\mathbf{a} = (a_1, a_2, \ldots, a_n)$, the minimal invariant one can build in number of terms is the orbit sum $\sum_{\mathrm{Orb}(\mathfrak{G})} (\mathbf{x}^{\mathbf{a}})$ of $\mathbf{x}$.

Let $\mathbb{K}$ a field, we denote by $\mathbb{K}[\mathbf{x}]^G$ the ring formed by all polynomials invariant under the action of G.

$$\mathbb{K}[\mathbf{x}]^G := \{P \in \mathbb{K}[\mathbf{x}] | \forall \sigma \in G : \sigma \cdot P = P\}.$$

For any subgroups G of $\mathfrak{S}_n$ and $\mathbb{K}$ a field of characteristic 0, a result due to Hilbert and Noether state that the ring of invariant $\mathbb{K}[\mathbf{x}]^G$ is a free module of rank $\frac{n!}{|G|}$ over the symmetric polynomials in the variable $\mathbf{x}$. Computing the invariant ring $\mathbb{K}[\mathbf{x}]^G$ consists essentially in building algorithmically an explicit family (called *secondary invariant polynomials*) of generators of this free module.

Searching the secondary invariant polynomials from orbit sum of monomials whose vector of exponents is *canonical* (instead of all monomials) produces a gain of complexity of $|G|$ if we assume that all orbits are of cardinality $|G|$. This assumption is obviously false; however, in practice, it seems to hold in average and up to a constant factor [Borie(2011)]).

In [Borie and Thiéry(2011)], the authors calculate the secondary invariants of the 61^{st} transitive group over 14 variables whose cardinality is 50803200. Using the *canonical* monomials, they managed to build a family of 28 irreducible secondary invariants deploying a set of 1716 secondary invariants. This computation is unreachable by Gröbner basis techniques.

5 Computing primitive invariants for a permutation group

5.1 Introduction

We now apply our generation strategy to this problem concerning effective Galois theory.

Problem 5.1 *Let n a positive integer and G a permutation group, subgroup of $\mathfrak{S}_n$. Let $\mathbb{K}$ be a field and $\mathbf{x} := x_1, \ldots, x_n$ be n formal variables. Find a polynomial $P \in \mathbb{K}[x_1, \ldots, x_n]$ such that*

$$\{\sigma \in \mathfrak{S}_n | \sigma \cdot P = P\} = G.$$

A such polynomial is called a primitive invariant *for G.*

Problem 5.1 (exposed in [Girstmair(1987)] and [Abdeljaouad(2000)]) consists in finding an invariant P under the action of G such that its stabilizer $Stab_{\mathfrak{S}_n}(P)$ in $\mathfrak{S}_n$ is equal to G itself. Solving this problem becomes difficult when we want to construct a *primitive invariant* of minimal degree or a *primitive invariant* with a minimal number of terms.

5.2 Primitive invariant of minimal degree

Algorithm 2 Primitive invariant using stabilizer refinement

Prerequisites :

• $IntegerVectorsModPermgroup$: module to enumerate orbit representatives;

• $stabilizer\_of\_orbit\_of(G, v)$: a function returning the permutation group which stabilizes the orbit of v under the action of the permutation group G.

Arguments:

• G: A permutation group, subgroup of $\mathfrak{S}_n$.

```
def minimal_primitive_invariant(G) :
    cumulateStab ← SymmetricGroup(degree(G))
    chain ← [[(0, 0, ..., 0), cumulateStab, cumulateStab]]
    if Cardinality(cumulateStab) == Cardinality(G) :
        return chain
    for v ∈ IntegerVectorsModPermgroup(G) :
        AutV ← stabilizer_of_orbit_of(G, v)
        Intersect ← cumulateStab ∩ AutV
        if Cardinality(Intersect) < Cardinality(cumulateStab) :
            chain ← chain ∪ [v, AutV, Intersect]
            cumulateStab ← Intersect
            if Cardinality(cumulateStab) == Cardinality(G) :
                return chain
```

5.3 Benchmarks

Algorithm 2 terminates in less than an hour for any subgroup of $\mathfrak{S}_{10}$. Even, it can calculate some primitive invariants for a lot of subgroups with degree between 10 and 20 while the literature only provides examples up to degree 7 or 8. Using the same computer, this benchmark just collects the average time in seconds of execution of Algorithm 2 by executing systematically the algorithm on transitive groups of degree n.

Degree of Groups	1	2	3	4	5	6	7	8	9
Computations time	0.008	0.064	0.104	0.160	0.208	0.393	0.537	2.364	27.093

This research was driven by computer exploration using the open-source mathematical software `Sage` [Stein et al.(2009)]. In particular, we perused its algebraic combinatorics features developed by the `Sage-Combinat` community [Sage-Combinat community(2008)], as well as its group theoretical features provided by `GAP` [GAP(1997)].

References

[Abdeljaouad(2000)] I. Abdeljaouad. *Théorie des Invariants et Applications à la Théorie de Galois effective*. PhD thesis, Université Paris 6, 2000.

[Bergeron(2009)] F. Bergeron. *Algebraic combinatorics and coinvariant spaces*. CMS Treatises in Mathematics. Canadian Mathematical Society, Ottawa, ON, 2009. ISBN 978-1-56881-324-0.

[Bergeron et al.(2011)Bergeron, Borie, and Thiéry] F. Bergeron, N. Borie, and N. M. Thiéry. Deformed diagonal harmonic polynomials for complex reflection groups. In *23rd International Conference on Formal Power Series and Algebraic Combinatorics (FPSAC 2011)*. 2011.

[Borie(2011)] N. Borie. *Calcul des invariants des groupes de permutations par transformée de Fourier*. PhD thesis, Laboratoire de Mathématiques, Université Paris Sud, 2011.

[Borie and Thiéry(2011)] N. Borie and N. M. Thiéry. An evaluation approach to computing invariants rings of permutation groups. In *Proceedings of MEGA 2011*, March 2011.

[Derksen and Kemper(2002)] H. Derksen and G. Kemper. *Computational invariant theory*. Springer-Verlag, Berlin, 2002. ISBN 3-540-43476-3.

[Faugère and Rahmany(2009)] J. Faugère and S. Rahmany. Solving systems of polynomial equations with symmetries using SAGBI-Gröbner bases. In *(ISSAC 2099)*, 2009.

[GAP(1997)] *GAP – Groups, Algorithms, and Programming*. The GAP Group, Lehrstuhl D für Mathematik, RWTH Aachen, Germany and SMCS, U. St. Andrews, Scotland, 1997.

[Gessel(1987)] I. M. Gessel. Enumerative applications of symmetric functions. In *Proceedings of the 17-th Séminaire Lotharingien, Publ. I.R.M.A. Strasbourg*, page 17, 1987.

[Girstmair(1987)] K. Girstmair. On invariant polynomials and their application in field theory. *Math. Comp.*, 48(178):781–797, 1987. ISSN 0025-5718. doi: 10.2307/2007843.

[Hartke and Radcliffe(2009)] S. G. Hartke and A. J. Radcliffe. McKay's canonical graph labeling algorithm. In *Communicating mathematics*, volume 479, pages 99–111. 2009.

[Hulpke(2005)] A. Hulpke. Constructing transitive permutation groups. *J. Symbolic Comput.*, 39(1): 1–30, 2005. ISSN 0747-7171. doi: 10.1016/j.jsc.2004.08.002.

[King(2007)] S. King. Fast Computation of Secondary Invariants. *Arxiv math/0701270*, 2007.

[MacMahon(2004)] P. A. MacMahon. *Combinatory analysis. Vol. I, II*. 2004. Reprint of ıt Combinatory analysis. Vol. I, II (1915, 1916).

[Sage-Combinat community(2008)] T. Sage-Combinat community. Sage-Combinat: enhancing Sage as a toolbox for computer exploration in algebraic combinatorics, 2008.

[Seress(2003)] Á. Seress. *Permutation group algorithms*, volume 152 of *Cambridge Tracts in Mathematics*. Cambridge University Press, Cambridge, 2003.

[Stein et al.(2009)] W. Stein et al. *Sage Mathematics Software (Version 3.3)*. The Sage Development Team, 2009. `http://www.sagemath.org`.