Upper Bounds and Probability Heuristic Estimates of the Number of r-Regular Families of Permutations

Pavol Kollár[1,*,†], Martin Mačaj[2,†]

[1]Department of Applied Informatics, Faculty of Mathematics, Physics and Informatics, Comenius University, Bratislava, Slovakia
[2]Department of Algebra and Geometry, Faculty of Mathematics, Physics and Informatics, Comenius University, Bratislava, Slovakia

Abstract

A sure-fire way of answering "How many of these objects are there?" is to generate them all and simply keeping a counter. However, if it so happens that the true count is about 10^{24}, this way of doing it becomes a "sure-fire way of getting nowhere". This is exactly what happens with so called r-regular families of permutations, which are a generalisation of the notion of transitivity in groups. They also measure how Cayley-like a vertex-transitive graph is. Therefore, another approach is required for this (and many other) cases. In this article, we shall describe a general-purpose algorithm that utilises complex cyclotomic numbers to give upper bounds to the counts of these families. Additionally, we present a probabilistic algorithm to give heuristic estimates of the true counts of these families.

Keywords

r-regular families, Generating functions, Fourier transformations, GAP

1. Introduction

As defined in [1], for any set M, an "r-regular family" of permutations is a set $\mathcal{F} \subseteq \mathbb{S}_M$, such that for every $x, y \in M$, there are exactly r permutations $\phi \in \mathcal{F}$, such that $\phi(x) = y$. Sometimes we will take $M = \{1, 2, 3, \ldots, n\}$, in which case we will use the standard shorthand $[n] := \{1, 2, 3, \ldots, n\}$. The r-regular family as an object is closely related to transitive groups, when their stabiliser is of size r. Indeed, such a group does follow the property of r-regular families and is therefore a special case of them. On the other hand, the r-regular families are combinatorial approximations of these transitive groups, as they themselves need not to follow the group axioms, only the "r regularity".

The authors of [1] investigated "how close to a Cayley graph a given vertex-transitive graph is", which is a question that has been asked many times in prior research in different variations. As such, there are multiple measures of this "closeness" notion. One such notion is to find the smallest transitive subgroup of $Aut(\Gamma)$. A different notion relies precisely on the aforementioned r-regular families, and also gets investigated in [1]. There, one seeks small r-regular families as subsets of the $Aut(\Gamma)$, the group of automorphisms of the given graph Γ. The paper [1] is the first instance we could find which mentions the use of r-regular families.

The simpler version of 1-regular family (or simply just *regular family*) was introduced much earlier in [2] already. In this paper, the author was looking at the well known issue of graph theory, that there are some vertex transitive graphs, which are not Cayley graphs, most famously the Petersen graph. Her method involved the construction of "quasi-Cayley graphs", which are tied to the existence of 1-regular families of automorphisms as follows: "a graph Γ is quasi-Cayley if and only if $Aut(\Gamma)$ contains a 1-regular family as a subset". This is similar to what Sabidussi did in [3] with Cayley graphs and their automorphism groups: "a graph Γ is Cayley if and only if $Aut(\Gamma)$ contains a subgroup, which acts regularly on $V(\Gamma)$".

With definitions laid out as we have, we can rephrase the result of Sabidussi as "a graph Γ is Cayley if and only if $Aut(\Gamma)$ contains a subgroup, which is also a 1-regular family (with respect to its action on $V(\Gamma)$)".

One year after the paper [1], in his bachelor thesis [4], the author investigated r-regular families from a computational perspective, specifically, by generating these families with the help of a computer.

In Section 2, we will summarise Kerák's main result from [4], which is relevant to our work, introduce the main problem we will be solving, and describe how generating functions and Fourier transformations can help us get upper bounds on the number of r-regular families. Continuing that, in Section 3, we talk about some of the implementation details of these methods to get said upper bounds. In Section 4, we briefly describe the method [4] used and expand on it with a randomised algorithm, which can be used as an estimation heuristic to approximate the number of r-regular families on n elements. Finally, in Section 5, we summarise our findings.

ITAT'24: Information technologies – Applications and Theory, September 20–24, 2024, Drienica, Slovakia
*Corresponding author.
†These authors contributed equally.
pavol.kollar@fmph.uniba.sk (P. Kollár);
martin.macaj@fmph.uniba.sk (M. Mačaj)

Table 1
Generated counts of r-regular families:

	$r=0$	$r=1$	$r=2$	$r=3$	$r=4$	$r=5$	$r=6$	$r=7$	$\cdots$
$n=1$	1	1	-	-	-	-	-	-	$\cdots$
$n=2$	1	1	-	-	-	-	-	-	$\cdots$
$n=3$	1	2	1	-	-	-	-	-	$\cdots$
$n=4$	1	24	255	640	255	24	1	-	$\cdots$
$n=5$	1	1344	11073216	NG	NG	NG	NG	NG	$\cdots$
$n=6$	1	1128960	NG	NG	NG	NG	NG	NG	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

2. Ground Work

2.1. Establishing notions

Our work picks up where [4] left off, where the author was generating the r-regular families with the help of a computer. One of his outputs was the Table 1 containing information on how many distinct r-regular families on n elements are there for some small r, n. Within this table, "NG" means that for those parameters, the families were only partially generated or not at all.

For the rest of the paper, we fix $n \in \mathbb{N}$ and also let $M = \{m_1, m_2, \ldots, m_n\}$ be a fixed set of n positive integers with $m_1 \leq m_2 \leq \ldots \leq m_n$. Borrowing some notation from [4], instead of writing down permutations of $\mathbb{S}_M$ with the cycle notation as is usual in algebra, e.g. $(1,2,4,6)$ or $(2,7,3)(5,11)$, we will write down all the numbers in one sequence into a list. In this list, the number k is in the p^{th} position in the list precisely when m_p is mapped to the number k by this permutation. Put another way, to each permutation $\phi \in \mathbb{S}_M$, we associate the list of numbers $L(\phi) = [\phi(m_1), \phi(m_2), \phi(m_3), \ldots, \phi(m_n)]$. For $M = [6]$ and $\phi = (1,2,4,6)$, this list will thus be $[2,4,3,6,5,1]$ and for $M = \{2,3,5,7,11,13\}$ and $\phi = (2,7,3)(5,11)$, the list will be $[7,2,11,3,5,13]$. When we talk about the elements of $\mathbb{S}_M$ from now on, we will mostly talk about the set of lists induced by the previous construction.

Observation 1. *Within this list association, an r-*regular family *of $\mathbb{S}_M$ is such a collection of the lists $\mathcal{F} \subseteq \mathbb{S}_M$, so that when the lists in $\mathcal{F}$ are aligned below one another, each column contains each $m \in M$ precisely r times.*

Later, we will want to talk about individual entries of these lists and for that reason, we shall access them via indices written with square brackets, i.e. $L(\phi)[i] = \phi(m_i)$. For us, the first entry has index 1, e.g. $[7,2,11,3,5,13][1] = 7, [2,4,3,6,5,1][4] = 6$.

Definition 1. *For a given non-negative integer r, let $R_r^M \subseteq \mathcal{P}(\mathbb{S}_M)$ be the set of r-regular families of $\mathbb{S}_M$. Also let $R^M = \bigcup_{r \geq 0} R_r^M$.*

Given any $r \leq (n-1)!$, we know that $\forall \mathcal{F} \in R_r^M : |\mathcal{F}| = nr$. This facilitates the fact that the only 0-regular set is the empty set and so $\left|R_0^M\right| = 1$ for *all* M regardless of its size. Since we have chosen $|M| = n$, we have that $|\mathbb{S}_M| = n!$, which means that for all $r > (n-1)!$, the set R_r^M is empty, so $\forall r > (n-1)! : \left|R_r^M\right| = 0$.

Our main effort for the remainder of this paper is to get upper bounds for the total number of r-regular families for $n = 5$. In Section 4 we will also describe a randomised algorithm that provides a heuristic estimate for the number of r-regular families on n elements are there in that specific case. We are also going to reference the values within the row for $n = 4$ in the Table 1 to check the correctness of our methods.

2.2. Generating functions

To bound the values of the numbers of r-regular families, we will use generating functions and their properties. To be able to use generating functions, we require a weight function on the subsets of $\mathbb{S}_M$, namely $W : \mathcal{P}(\mathbb{S}_M) \to \mathbb{N}$. Instead of producing a weight for every subset, we assign each *permutation* a weight with the function $w : \mathbb{S}_M \to \mathbb{N}$ and the total weight of a subset of $\mathbb{S}_M$ shall simply be the sum of individual weights.

The weight function of single permutations $w : \mathbb{S}_M \to \mathbb{N}$ will depend on a chosen list $\mathbf{w} = [w_1, \ldots, w_n]$ of weights (and also the chosen set M) and is defined by $w(\phi) = \sum_{i=1}^{n} w_i \cdot L(\phi)[i]$. In a sense, we really should call this weight function $w_{\mathbf{w}}$, but we will omit it in this section. For example, if $M = [4]$ and $\mathbf{w} = [1,2,5,15]$, for the identity permutation we have $w(e) = 1 \cdot 1 + 2 \cdot 2 + 5 \cdot 3 + 15 \cdot 4 = 80$. For $\phi = (1,2)$, we have $L(\phi) = [2,1,3,4]$ and $w(\phi) = 79$, and for $\phi = (1,4,3)$, we have $L(\phi) = [4,2,1,3]$ and $w(\phi) = 62$.

As mentioned before, the weight function for sets of permutations $W : \mathcal{P}(\mathbb{S}_M) \to \mathbb{N}$, will be the sum of the individual permutation weights, i.e., $W(X) = \sum_{\phi \in X} w(\phi)$. For any $A \subseteq \mathbb{N}$, we can ask for the set $C_A = \{F \subseteq \mathbb{S}_M | W(F) \in A\}$. Ideally we want to construct the function W (that is, choose the list $\mathbf{w}$ as well as the set M in a clever way) so that there exists an A

with $\mathcal{F} \in R_r^M \Leftrightarrow W(F) \in A$. However, ensuring both implications hold *while also* ensuring that he entries of M and $\mathbf{w}$ are sufficiently small is highly non-trivial (for $M, \mathbf{w}$ big, it is not so difficult to come up with examples), so we will relax the requirement and demand that only $\mathcal{F} \in R_r^M \Rightarrow W(F) \in A$ holds. This is why our algorithms will, in general, produce upper bounds for those counts, and occasionally produce the exact values.

Continuing the simple example above with $M = [4]$, one can see that for any r-regular family $\mathcal{F}$, each weight is multiplied r times by each $m_i \in M$, so $W(\mathcal{F}) = (1+2+5+15) \cdot r \cdot (1+2+3+4) = 230r$. In general we have the following lemma.

Lemma 1. *Fix $\mathbf{w} = [w_1, \ldots, w_n]$, and let $\mathcal{F} \subseteq \mathbb{S}_M$ be an r-regular family. Then $W(\mathcal{F}) = r \cdot \sum_{m_i \in M} m_i \cdot \sum_{w_i \in \mathbf{w}} w_i$. Therefore, with $d = \sum_{m_i \in M} m_i \cdot \sum_{w_i \in \mathbf{w}} w_i$ and $A = \{k \cdot d | k \in \mathbb{N}\}$, the condition $\mathcal{F} \in R_r^M \Rightarrow W(\mathcal{F}) \in A$ is satisfied for all r, in particular $\mathcal{F} \in R^M \Rightarrow W(\mathcal{F}) \in A$.*

Proof. Let $\mathcal{F}$ be an r-regular family. By definition,

$$W(\mathcal{F}) = \sum_{\phi \in \mathcal{F}} w(\phi) = \sum_{\phi \in \mathcal{F}} \sum_{w_i \in \mathbf{w}} w_i \cdot L(\phi)[i],$$

where the $L(\phi)[i]$ are elements of M. This is a finite sum and hence we can swap the order of summations:

$$W(\mathcal{F}) = \ldots = \sum_{w_i \in \mathbf{w}} w_i \cdot \sum_{\phi \in \mathcal{F}} L(\phi)[i].$$

Because $\mathcal{F}$ is r-regular, together over all i and all ϕ, $L(\phi)[i]$ equals each $m_j \in M$ precisely r times:

$$W(\mathcal{F}) = \ldots = \sum_{w_i \in \mathbf{w}} w_i \cdot \sum_{m_j \in M} r \cdot m_j.$$

To finish the proof of the claim, pull r out of both sums, since it is independent of the sums, and substitute in the defined d:

$$W(\mathcal{F}) = \ldots = r \cdot \sum_{w_i \in \mathbf{w}} w_i \cdot \sum_{m_j \in M} m_j = r \cdot d.$$

This means that d divides $W(\mathcal{F})$, so $W(\mathcal{F}) \in A$. □

Said another way that is more usual in the context of generating functions, if $a_k = |\{F \subseteq \mathbb{S}_M : W(F) = k\}|$, then the following bound is a direct consequence of the previous lemma:

$$\left|R^M\right| \leq \sum_{r \in \mathbb{N}} a_{rd} \tag{1}$$

We take $g(x)$ to be the generating function for this sequence of numbers, that is, $g(x) = \sum_{n \in \mathbb{N}} a_n \cdot x^n$. By standard theory of generating functions (see e.g. [5]), we know this generating function g is also equal to $g(x) = \prod_{\phi \in \mathbb{S}_M} (1 + x^{w(\phi)})$, creating the following essential equation:

$$\sum_{k \in \mathbb{N}} a_k \cdot x^k = g(x) = \prod_{\phi \in \mathbb{S}_M} (1 + x^{w(\phi)}) \tag{2}$$

Utilising a standard trick of Fourier transformations, let ζ a primitive d^{th} root of unity. Then to get the bounding sum $\sum_{r \in \mathbb{N}} a_{rd}$, we can average the values of $g(x)$ at all the d^{th} roots of unity, that is:

$$\sum_{r \in \mathbb{N}} a_{rd} = \frac{1}{d} \cdot \sum_{i=1}^{d} g(\zeta^i) \tag{3}$$

Combining Equation 1 and Equation 3, we get

$$\left|R^M\right| \leq \sum_{r \in \mathbb{N}} a_{rd} = \frac{1}{d} \cdot \sum_{i=1}^{d} g(\zeta^i) \tag{4}$$

The evaluation of this average requires the evaluations of g on the right hand side. Those values depend on the weight function w, which in turn depends on the particular choice of M and $\mathbf{w}$. This constitutes the first algorithm, which accepts inputs $M, \mathbf{w}$ and outputs the upper bound from Equation 4, see Section 3.1 for implementation.

2.3. Generating functions in more than one variable

In the previous section, we transformed a permutation ϕ into its weight $w(\phi) = \sum_{w_i \in \mathbf{w}} w_i \cdot L(\phi)[i]$ and we transformed a permutation set $F \subseteq \mathbb{S}_M$ into $W(F) = \sum_{\phi \in F} w(\phi)$. Enconding each set of permutations into a single number caused us to lose a lot of information about said set of permutations. Instead, let us assign the "weight" of the permutation to be the permutation itself in the list notation: $w(\phi) = L(\phi)$. Treating these lists of integers as vectors for the purposes of addition and scalar multiplication, we may now define $W(F) = \sum_{\phi \in F} w(\phi)$ for any $F \subseteq \mathbb{S}_M$. This way, we retain more information about F.

Lemma 2. *Let $\mathcal{F} \subseteq \mathbb{S}_M$ be an r-regular family. Then $W(\mathcal{F}) = r \cdot \sum_{m \in M} m \cdot [1, 1, \ldots, 1]$, where the vector has $|M| = n$ ones. With $\mathbf{d} = \sum_{m \in M} m \cdot [1, 1, \ldots, 1]$ and $A = \{k \cdot \mathbf{d} | k \in \mathbb{N}\}$, the condition $\mathcal{F} \in R_r^M \Rightarrow W(\mathcal{F}) \in A$ is satisfied for all r, in particular $\mathcal{F} \in R^M \Rightarrow W(\mathcal{F}) \in A$.*

Proof. The proof is analogous to the proof of the one-dimensional case from the previous subsection. Each position within the lists gets each number $m \in M$ precisely r times in the expression for $W(\mathcal{F})$, when $\mathcal{F}$ is an r-regular family. □

To avoid the tiny writing of double indexing in a multivariable coefficient, let $\mathbf{I} = [i_1, i_2, \ldots, i_n]$. Denoting by $b_{\mathbf{I}} = |\{F \subseteq \mathbb{S}_M : W(F) = \mathbf{I}\}|$, Lemma 2 tells us that

$$\left|R^M\right| \leq \sum_{r \in \mathbb{N}} b_{r \cdot \mathbf{d}} \tag{5}$$

Once again, take $G(\mathbf{x}) = G(x_1, x_2, \ldots, x_n)$ to be the multivariate generating function for the values of b:

$$G(\mathbf{x}) = \sum_{\mathbf{I} \in \mathbb{N}^n} b_{\mathbf{I}} \cdot x_1^{i_1} \cdot x_2^{i_2} \cdot \ldots \cdot x_n^{i_n}.$$

Just like in the one dimensional case, G can be written as a product, which we record in the next equation:

$$\begin{aligned} \sum_{\mathbf{I} \in \mathbb{N}^n} b_{\mathbf{I}} \cdot x_1^{i_1} \cdot x_2^{i_2} \cdot \ldots \cdot x_n^{i_n} = \\ = G(x_1, x_2, \ldots, x_n) = \\ = \prod_{\phi \in \mathbb{S}_M} (1 + \prod_{i=1}^{n} x_i^{w(\phi)[i]}) \end{aligned} \tag{6}$$

Let $s = \sum_{m_i \in M} m_i$, which means that $\mathbf{d}$ can be written as $[s, s, \ldots, s]$. Here too, we want to use the trick of Fourier transformations. In this case we can start with the observation that:

$$\frac{1}{s^n} \cdot \sum_{\mathbf{I} \in \mathbb{Z}_s^n} G(\zeta^{i_1}, \ldots, \zeta^{i_n}) = \sum_{\mathbf{I} \equiv \mathbf{0} \pmod{s}} b_{\mathbf{I}} \tag{7}$$

Here, the second sum is over all vector-indices, whose each coordinate is a multiple of s. Clearly, the sum on the right-hand-side includes all coefficients with indices $k \cdot \mathbf{d}$, therefore we have the following upper bound as a consequence:

$$\left|R^M\right| \leq \sum_{r \in \mathbb{N}} b_{r \cdot \mathbf{d}} \leq \frac{1}{s^n} \cdot \sum_{\mathbf{I} \in \mathbb{Z}_s^n} G(\zeta^{i_1}, \ldots, \zeta^{i_n}) \tag{8}$$

The algorithm implementing this approach and its optimisations is described in Subsection 3.3.

3. Algorithmic Upper Bounds

3.1. The basic algorithm

Let us return to the ideas from Section 2.2 and create an algorithm based on them. For the Subsection 3.1, fix $\mathbf{w} = [w_1, \ldots, w_n]$ and also M. Furthermore, as before, let $d = \sum_{m_i \in M} m_i \cdot \sum_{w_i \in \mathbf{w}} w_i$. At the end of Subsection 2.2, we ended up with the bounding inequality described in 4. Our algorithm will thus, for every ζ^i, evaluate the polynomial g at that value and average these results. We remark that by fixing M, $\mathbf{w}$ (which will be the inputs of the algorithm), we are also fixing the value d as well as the polynomial g. This means that the values $g(\zeta^i)$ are unambiguously determined by the input.

To evaluate these products, we will need a program that can handle addition and multiplication of these kinds of complex numbers - formally known as *cyclotomic numbers* - with exact precision. Fortunately, the GAP system [6] has rich support for arithmetic with complex roots of unity, cyclotomic numbers, their sums, their products (and of much more). This - alongside our previous experience with the system - is why GAP is our software of choice for these computations, the pseudocode for which we present in Algorithm 1.

Algorithm 1 One variable bounding algorithm

Input: M, $\mathbf{w}$
Output: Upper bound for the size $\left|R^M\right|$
$powers\_of\_x \leftarrow [\,]$
for $perm \in \mathbb{S}_M$ **do** ▷ Here, $perm$ is taken as a list.
  $powers\_of\_x$.append($\sum_{i=1}^{n} perm[i] \cdot w_i$)
end for
$d \leftarrow \sum_{m_i \in M} m_i \cdot \sum_{w_i \in \mathbf{w}} w_i$
$\zeta \leftarrow E(d)$ ▷ Primitive root of unity.
$accumulator \leftarrow 0$
for $1 \leq i \leq d$ **do**
  $product \leftarrow 1$
  for $power \in powers\_of\_x$ **do**
    $product \leftarrow (1 + \zeta^{i \cdot power}) \cdot product$
  end for
  $accumulator \leftarrow accumulator + product$
end for
return $accumulator/d$ ▷ Always an integer.

Testing this algorithm on various inputs, we recognised that to get a meaningful decrease of the computed upper bound, we needed to substantially increase the value of d. Because $g(\zeta^i) = \prod_{\phi \in \mathbb{S}_M} (1 + \zeta^{i \cdot w(\phi)})$, the computation of each $g(\zeta^i)$ is done in linear time with respect to the order of ζ (which is d), and therefore the runtime complexity is $O(n! \cdot d^2)$. Coupled with the fact that we had to substantially increase d to get a significant improvement, we stopped using this exact approach, even if the further algorithms share the same core idea. Still, we include the description of the algorithm here to illustrate this core idea. Later, in the results, we will highlight where the particular approaches had their computational limits.

3.2. Long lists of coefficients

To make matters for the previous algorithm worse, even if we could reduce the time complexity of the algorithm down, we still have a memory overflow waiting to happen. In fact, one of our intermediate algorithm iterations had time complexity just $O(n! \cdot d)$, because instead of

evaluating d different values $g(\zeta^i)$, this algorithm expanded the product form of g symbolically. This is because what the computation *actually* needs is the sum of certain polynomial coefficients of g - specifically those whose index is $\equiv 0 \pmod{d}$. To perform this expansion of $g(x)$, whose product form we established in Equation (2), we represent each intermediate step as a long list of coefficients, in which each position is the sum of all coefficients, whose indices have the same remainder $\pmod{d}$. At the end, the result we seek is then simply the 0^{th} coefficient of this list.

While this provided a significant time-improvement, given that we only "evaluated" the polynomial once, the memory requirement was roughly $O(d)$, i.e., an approximately linear amount of memory is required for this computation. On our laptop, even with tricks of the "meet in the middle" character, choosing $d \approx 1.2 \cdot 10^8$ was roughly the limit of its memory capabilities. A stronger machine could be employed for the task, but that too does not solve the problem. This is because the memory problem is, in a sense, unavoidable. Once we start to dabble with cyclotomic numbers that we need to represent exactly, and we need to have $\varphi(d)$, where φ is the Euler totient function, coefficients to represent those. Choosing d to have many distinct prime factors seemed to yield worse upper bounds than when d had only a few of those and then $\varphi(d)$ is still $O(d)$, so we cannot escape the issue, at least not in this way.

3.3. Thinking in more dimensions

As we have argued, the basic algorithms have large memory requirements, some with bad runtimes in relation to the results we were getting from them. Our third and most recently used method for this problem seeks to improve this via generating polynomials of many variables, as was discussed and introduced in Section 2.3.

In what follows, M is still the set $\{m_1, m_2, \ldots, m_n\}$ and let $s = \sum_{m_i \in M} m_i$. To simplify the optimisation arguments that follow in the later part of the section, assume that s is a prime number bigger than n, hence n and s are coprime. Finally, let ζ be a primitive s^{th} root of unity, just as in the Section 2.3. In that section, we arrived at the inequality in (8), which bounded the total number of all r-regular families on n elements by the expression:

$$\frac{1}{s^n} \cdot \sum_{\mathbf{I} \in \mathbb{Z}_s^n} G(\zeta^{i_1}, \ldots, \zeta^{i_n}).$$

This means that all our algorithm needs to do is to evaluate the multinomial G on all possible vectors of roots of unity and average the result. However, this is outrageously slow. Notice that for that to happen, we require s^n evaluations of G. Moreover, as we discussed in Section 3.1, the evaluations themselves are linear in s in terms of time, so the total time complexity is roughly $O(n! \cdot s^{n+1})$. For just $n = 5$ and $s = 127$, that causes the algorithm to run for about 15 hours, while giving a weak bound.

Since the evaluations of G themselves cannot be easily sped up, let us try to decrease the number of evaluations. We will choose some properties of the set M that will lead to optimisation of our algorithm's speed. With the claims that follow, we will prove that we can reduce the complexity down to only $O(s^{n-1})$. Also, to simplify the further expressions, we will write $G[i_1, \ldots, i_n]$ instead of $G(\zeta^{i_1}, \ldots, \zeta^{i_n})$. Because of the equality in (6), it easily follows that:

Lemma 3. *For any $[i_1, \ldots, i_n] \in \mathbb{Z}_s^n$ and any $\varphi \in \mathbb{S}_n$ we have*

$$G[i_1, \ldots, i_n] = G[i_{\varphi(1)}, \ldots, i_{\varphi(n)}].$$

For fixed n, the next lemma will gain us a much more significant speed-up:

Lemma 4. *Adding 1 to all the exponents of the roots of unity does not change the value of G, i.e.:*

$$G[i_1, \ldots, i_n] = G[i_1 + 1, \ldots, i_n + 1].$$

Proof. Focus on the specific product $x_1^{m_1} \cdot x_2^{m_2} \cdot \ldots \cdot x_n^{m_n}$ within G. Evaluating it at the powers of ζ from the left-hand-side, we of course get another power of ζ. The value of that power is

$$\sum_{k=1}^{n} i_k \cdot m_k.$$

On the other hand, evaluating that product on the right-hand-side gives us (again omitting the ζ itself and writing down just the power):

$$\sum_{k=1}^{n} (i_k + 1) \cdot m_k = \sum_{k=1}^{n} i_k \cdot m_k + s.$$

These two expressions only differ by the $+s$ at the end and both of these expressions represent powers of ζ. Since ζ is a primitive s^{th} root of unity, the $+s$ makes no difference to the product.

As such, this product within G has the same value on the inputs we started with and, analogously, so will every other product in G. This proves our claim. □

We can iteratively repeat this argument to get the following equality as an immediate corollary

$$G[i_1, \ldots, i_n] = G[i_1 + k, \ldots, i_n + k] \tag{9}$$

that holds for all $1 \leq k \leq s$, with $k = s$ being the trivial case, as exponents of ζ operate $\pmod{s}$.

Picking s to be prime is essential for the second big optimisation. Before we get to that optimisation, we mention a bit about how GAP internally stores cyclotomic numbers. We touched on this earlier in Section 3.2, and we will be more specific here. For prime s the cyclotomic field $\mathbb{Q}(\zeta)$ is an $s-1$ dimensional linear space over $\mathbb{Q}$. Instead of "the standard basis" $1, \zeta, \ldots, \zeta^{s-2}$, GAP uses the basis $\zeta, \zeta^2, \ldots, \zeta^{s-1}$, that is, each element of $\mathbb{Q}(\zeta)$ is expressed as $c_1\zeta + c_2\zeta^2, \ldots + c_{s-1}\zeta^{s-1}$ and it is internally represented as vector $[c_1, c_2 \ldots, c_{s-1}]$. In GAP, it is possible to obtain this vector via the function `CoeffsCyc`. Let us note that any rational number q is represented by the vector $[-q, -q, \ldots, -q]$.

Let us also note that this basis is invariant under the Galois group of $\mathbb{Q}(\zeta)$ which consists of automorphisms ψ_k, which map $1 \mapsto 1, \zeta \mapsto \zeta^k$ and act on the roots of unity as a permutation for all $1 \leq k < s$. Note that ψ_1 is the identity map. With all the notation and knowledge of this subsection, we are ready to state the next optimisation claim:

Lemma 5. *Let $[i_1, \ldots, i_n] \in \mathbb{Z}_s^n$ and let $G[i_1, \ldots, i_n] = \sum_{j=1}^{s-1} c_j\zeta^j$. Then*

$$\sum_{k=1}^{s-1} G[ki_1, \ldots, ki_n] = \sum_{k=1}^{s-1} \psi_k(G[i_1, \ldots, i_n]) = -\sum_{j=1}^{s-1} c_j \tag{10}$$

Proof. Let us note that for any $[a_1, \ldots, a_n] \in \mathbb{Z}^n$ we have $\psi_k((\zeta^{i_1})^{a_1} \cdot \ldots \cdot (\zeta^{i_n})^{a_n}) = (\zeta^{ki_1})^{a_1} \cdot \ldots \cdot (\zeta^{ki_n})^{a_n}$. Therefore, by (6), we have $\psi_k(G[i_1, \ldots, i_n]) = G[ki_1, \ldots, ki_n]$, which is the first claimed equation.

As s is prime, Galois group of $\mathbb{Q}(\zeta)$ acts regularly on $\{\zeta, \ldots, \zeta^{s-1}\}$. Therefore $\sum_{k=1}^{s-1} \psi_k(G[i_1, \ldots, i_n]) = \sum_{i=1}^{s-1} \left(\sum_{j=1}^{s-1} c_j\right) \zeta^i$. Because $\sum_{i=1}^{s-1} \zeta^i = -1$, rearranging the last double-sum yields $-\sum_{j=1}^{s-1} c_j$. □

Previous lemmas describe how corresponding actions of $\mathbb{S}_n$, $\mathbb{Z}_s$, and $\mathbb{Z}_s^*$ on the set $\mathbb{Z}_s^n$ modify evaluation of $G(\mathbf{X})$. Combining these lemmas we obtain information about the action of group $\Gamma = \mathbb{S}_n \times \mathbb{Z}_s \times \mathbb{Z}_s^*$ on the set $\mathbb{Z}_s^n$ and its behaviour with the evaluations of $G(\mathbf{X})$.

Theorem 1. *Let $\mathbf{I} = [i_1, \ldots, i_n] \in \mathbb{Z}_s^n$ and let $G[i_1, \ldots, i_n] = c_1\zeta + \ldots + c_{s-1}\zeta^{s-1}$. Then there exists a number $B_{\mathbf{I}}$ which depends only on (the conjugacy class of) the stabilizer $\Gamma_{\mathbf{I}}$ such that*

$$\sum_{\mathbf{J} \in \mathbf{I}^\Gamma} G[j_1, \ldots, j_n] = B_{\mathbf{I}}(-c_1 - \ldots - c_{s-1}) \tag{11}$$

The exact correspondence between $B_{\mathbf{I}}$ and $\Gamma_{\mathbf{I}}$ is too complicated to present here. Let us just point out that (in the case that s is prime and greater than n) for $\mathbf{I} = [0, \ldots, 0]$ we have $B_{\mathbf{I}} = 1$ and for $\Gamma_{\mathbf{I}} = \{e_\Gamma\}$ we have $B_{\mathbf{I}} = s \cdot n!$.

A standard way to utilize the Theorem 1 is to choose canonical representatives of orbits of Γ. Then one needs to derive efficient ways of determining whether given vector $\mathbf{I}$ is the representative of its orbit as well as how to compute $B_{\mathbf{I}}$. One possible way to pick the canonical representative is to choose those $\mathbf{I}$'s, which are lexicographically smallest as vectors. For this choice of representatives, we have the following restrictions:

- A canonical representative has non-decreasing elements, because of Lemma 3.
- Thanks to Lemma 4, the representative has sum of its coefficients $\equiv 0 \pmod{s}$.
- As a consequence of Theorem 1, the first non-zero coordinate of a canonical representative is 1.

However not all such vectors are representatives of their orbits and the algorithm needs to be able to determine which is "the representative". For example, if $M = [1, 2, 3, 4, 7]$ and $s = 17$ both $[0, 1, 2, 3, 11]$ and $[0, 1, 9, 10, 14]$ belong to the same orbit of Γ.

The pseudocode for this improved algorithm is in Algorithm 2. For full implementation details of this algorithm, see our Github repository [7].

4. Heuristic Counting with Backtracking and Probability

Let us return to the work of Kerák [4] and his recursive algorithm to generate all r-regular families on n elements. In what follows, assume $n = 5$ is fixed and, since the specific choice of M does not change the number of r-regular families, let $M = [5] = \{1, 2, 3, 4, 5\}$. On top of that, partition the set $\mathbb{S}_M$ into five sets $S_1, \ldots, S_5$, so that the set S_i contains precisely those permutations $\phi \in \mathbb{S}_M$, such that $\phi(1) = i$. We remark that this makes all S_i have equal size: $|S_1| = \ldots = |S_5| = (5-1)! = 24$.

Secondly, with this partition in place, we can see that any r-regular family $\mathcal{F} \subseteq \mathbb{S}_M$ must arise as a union of r-sized subsets $T_i \subseteq S_i$, such that all the other positions contain each i precisely r times too.

That is precisely what Kerák's algorithm does - recursively trying out all subsets $T_i \subseteq S_i$ of size r, exiting early if any position overflows , i.e., if any position has more than r of any number $i \in \{1, 2, 3, 4, 5\}$. Once a $T_i \subseteq S_i$ has been found to not overflow any position with any number, his algorithm proceeds recursing in a depth-first manner, until all possible combinations of subsets of the S_i's have been tested. After all these families have been generated, counting them up is easy.

This approach is correct and it generates everything, but the caveat here is that there are too many possibilities.

Algorithm 2 Multivariate bounding

Input: M
Output: Upper bound for the size $\left|R^M\right|$

$n \leftarrow \texttt{Length}(M)$
$s \leftarrow \texttt{Sum}(M)$ ▷ We could test s for primality.
$G \leftarrow ComputeGeneratingFunction(\,)$ ▷ Helper function.
$BigSum \leftarrow 0$
for $\mathbf{I} \in \mathbb{Z}_s^n$ **do**
  if $\texttt{Sum}(I) \neq 0 \pmod s$ **then**
    continue ▷ Not a canonical representative.
  end if
  if $\texttt{not}(\texttt{IsSortedList}(\mathbf{I}))$ **then**
    continue ▷ Not a canonical representative.
  end if
  if $\texttt{First}(\mathbf{I}, x \to x > 0) \neq 1$ **then**
    continue ▷ Not a canonical representative.
  end if
  $IsMinimal \leftarrow \texttt{true}$
  for $k \in \texttt{Difference}(\mathbf{I}, [0])$ **do**
    $\mathbf{J} \leftarrow \mathbf{I}/k \pmod s$
    if $\mathbf{J} < \mathbf{I}$ **then**
      $IsMinimal \leftarrow \texttt{false}$
      break ▷ Not a canonical representative.
    end if
  end for
  if $IsMinimal$ **then**
    $B_{\mathbf{I}} \leftarrow ComputeB_{\mathbf{I}}(\mathbf{I})$ ▷ Helper function.
    $BigSum \leftarrow BigSum + B_{\mathbf{I}} \cdot \texttt{Sum}(\texttt{CoeffsCyc}(G[\mathbf{I}], s))$
  end if
end for
return $BigSum/s^n$

Even for 2-regular families on 5 elements, each S_i has 24 permutations, which means that at each level, we have $\binom{24}{2} = 276$ choices to pick from. Doing that on 5 different levels, the total amount of possibilities is $276^5 = 1601568101376$, which is starting to be too much for a laptop. Even for a more powerful machine, it quickly gets unruly and completely infeasible for families of higher regularity - for example the naive strategy for 8-regular families on 5 elements has the amount of possibilities on the order of $\approx 2 \cdot 10^{29}$.

To simplify matters of memory usage, let us use this algorithm to only count the number of the families with this algorithm, not fully generate them all. This means that at the bottom level, if we confirm that the chosen set is an r-regular family, we just increment a global counter. Since our tree of possibilities is too big, let us assume that "it will look about the same in every branch".

This is not an outrageous assumption as we are studying regular sets of permutations, which are quite symmetric. Still, providing concrete bounds for the error this creates is something we have not done, so it cannot be said that this algorithm is much more than a method for obtaining a heuristic estimation of the true counts of r-regular families on n elements.

Table 2
Assorted results of the heuristic counting algorithm:

Sum of all counts exactly	Scientific notation
1344808311315380068372992	$1.345 \cdot 10^{24}$
1687144238754283380215130	$1.687 \cdot 10^{24}$
4952098573211019632607892	$4.952 \cdot 10^{24}$
3192278092063163925627380	$3.192 \cdot 10^{24}$
1072241417789838192913856 6	$1.072 \cdot 10^{25}$
5578509063361797937216054	$5.579 \cdot 10^{24}$
2047532970827157893748616	$2.048 \cdot 10^{24}$
Average of the trials: (rounded)	
4217826489633026395275233	$4.218 \cdot 10^{24}$

The trick is to only explore a handful of the branches into full depth, thus substantially reducing the number of explored nodes. Specifically, we will assign each recursion depth a probability, with which we let a branch deeper. We keep track, for each of the levels, how many branches *reached* the level, and how many branches were *let through*. At the end, thanks to our "all branches look the same" assumption, we can just divide the final count by the "numbers of branches let through" and multiply by the "numbers of branches reaching that level".

The probabilities for recursion depths for $n = 5$ and $r \in \{2, 3, 4, \ldots, 12\}$ were determined by experimentation, so that the expected number of branches that are let through on each recursion depth is between 20 and 200. This lets us get a non-trivial sample of the decision tree without making the algorithm absurdly long. The eleven instances usually run to completion in 7-8 hours, depending on the random branches that are picked by the algorithm.

We experimented with this algorithm, when $n = 4$, where the actual count of all r-regular families is 1200. On the vast majority of the trials, the algorithm outputs a number between 1150-1250, so we are reasonably confident that the algorithm does not give wildly incorrect results. The average of the results of the last couple of recorded trials is 1202. At the time of writing, we ran the algorithm for $n = 5$ seven times, and those results are in the Table 2. We also include the average of the trials. We considered excluding the largest and the smallest value from the average, but because we only have 7 data points until now, we decided against it.

For more detailed results, see the table in our Github repository [7]. We remark that in $n = 5$ case, one does not need to go past $r = 12$, even though r-regular families exist up to $r = 24$, because for each r-regular family

Algorithm 3 Probabilistic backtracking algorithm

```
counter ← 0
depth_probabilities ← [350, 2000, 3000, 200, 1]                    ▷ Chosen for n = 5, r = 4.
depths_let_forward, depths_reached ← [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]
procedure RecursiveBacktrack(partial, depth)
    if depth = n then                                   ▷ End of recursion and it has been checked already.
        counter ← counter + 1
    else
        for new_perms ∈ r-sized subsets of S_i do       ▷ The sets S_i have been created already.
            new_partial ← new_partial ∪ new_perms
            if ∃x, y ∈ [n], ∃T ⊆ new_partial : |T| > r ∧ ∀φ ∈ T : φ(x) = y then   ▷ Check overflows.
                continue                                ▷ Too many permutations mapping x to y. Try another one.
            else
                depths_reached[depth] ← depths_reached[depth] + 1
                if randint(1, depth_probabilities[depth]) = 1 then
                                ▷ Going further deep in the recursion with probability 1/depth_probabilities[depth].
                    depths_let_forward[depth] ← depths_let_forward[depth] + 1
                    RecursiveBacktrack(new_partial, depth + 1)
                end if
            end if
        end for
    end if
end procedure
RecursiveBacktrack({}, 0)                               ▷ To start the recursive algorithm off.
for (reached, let_forward) ∈ zip(depths_reached, depths_let_forward) do
                                ▷ Python's zip allows us to iterate through two lists in sync.
    counter ← counter · reached/let_forward
end for
return round(counter)                                   ▷ To have an integral result.
```

$\mathcal{F} \subseteq \mathbb{S}_M$, its complement $\mathcal{G} = \mathbb{S}_M \setminus \mathcal{F}$ is $(n-1)! - r$-regular. This means, that after 12, whatever the true counts were in the first half will also be in the second half, reflected about the column $r = 12$. Hence, the total number of r-regular families that exist on 5 elements is then calculated as double of all the numbers, except for the $r = 12$ case, that should be counted only once.

5. Conclusion

In this paper, we described methods of both getting upper bounds for the number of r-regular families on n elements as well as a method to get a heuristic estimation of the true counts. We checked our work for $n = 4$ elements, where to get a fast enough version of the algorithm 2, we did not need to use Theorem 1, so s did not need to be prime, just coprime to n. The bounds here quickly converge to the true count of 1200 (see Table 3) and the estimation heuristic also gets really close to this number (see the discussion in Section 4).

For the case $n = 5$, one can immediately see that an r-regular family has size divisible by 5. Therefore, the number of subsets of a set with 120 elements, whose sizes are disivible by 5 constitute a trivial (and very excessive!) bound for the total number of r-regular families:

$$\left|R^M\right| \leq \binom{120}{0} + \binom{120}{5} + \ldots + \binom{120}{120} \approx 2.658 \cdot 10^{35}.$$

The best bound the basic algorithm could produce before running out of memory was $107574405772195673480227709 30 \approx 1.076 \cdot 10^{28}$. After that, we switched to the algorithm using the multivariate generating function and we include a few of the output results in the table too. The lowest upper bound we obtained at the time of writing is $426388047537084351080035 6 \approx 4.264 \cdot 10^{24}$. Coupled with the estimation heuristic makes us fairly confident the upper bounds are approaching the true count.

Let us also contrast our method to the naive approach of "test every subset of $\mathbb{S}_5$ and check r-regularity". We know there are 2^{120} subsets of $\mathbb{S}_5$ and let us say we could check 2^{30} of them in a second (this is approaching roughly the clock speed of a modern computer and is probably a vast exaggeration of the machine's capabilities) - this approach would thus take 2^{90} seconds, roughly $3 \cdot 10^9$ ages of the universe.

Table 3
Assorted results of the bounding multivariate algorithm for $n = 4$:

m_1	m_2	m_3	m_4	$s = \sum_i m_i$	Time (ms)	Bound
1	2	4	8	15	5	5112
1	4	16	64	85	1268	1200
1	6	36	216	259	85609	1200
1	8	64	512	585	1566917	1200

Table 4
Assorted results of the multivariate algorithm for $n = 5$:

m_1	m_2	m_3	m_4	m_5	$s = \sum_i m_i$	Time (ms)	Bound
Trivial bound					-	0	265845599161775836797571384326161716
One-dimensional method					-	?	22669475893267033898649677785186
1	2	4	8	16	31	34	14393045699934445160465310000316
1	2	5	11	24	43	99	38879946384473700999015562359 6
Long list of coefficients method					-	$\sim$ 1800000	10757440577219567348022770930
1	3	9	27	81	127	4925	51096286946845957907766077 56
16	24	36	54	93	223	48255	54046552791883084789276 4076
1	4	16	64	592	677	4561658	12647340576853621194664376
1	5	27	141	733	907	16534644	62481179097119763191097 56
1	5	25	125	883	1039	28784230	53488805614589067966057 66
1	5	25	125	1403	1559	$\sim$ 85384605	42638804753708435108003 56
1	7	49	343	2401	2801	?	39269853921782910583211 16

On the other hand, for reasons not mentioned in this paper, choosing $M = \{1, 25, 625, 15625, 390625\}$ is sufficient to get the precise value of $\left|R^M\right|$. Here, $s = 406901$, and the algorithm evaluates roughly (in fact less than) s^3 points of the multinomial G, which means roughly $6.737 \cdot 10^{16}$ evaluations. Pesimistically, let us say that one G evaluation takes a minute, which would still "only take" about $128, 177, 239, 751$ years, which is 9.3 ages of the universe. All that is to say that the competition is not even close.

However, the $n = 6$ case is tricky. For one, because the upper bounding algorithm has time complexity $O(s^{n-1})$, raising n increases the runtime drastically. To get a meaningful bound, one will probably need to substantially improve this algorithm further. The estimation algorithm too will not work too well, because the branching factors in the probabilistic backtracking themselves get way too big, e.g. $\binom{120}{60} \approx 9.661 \cdot 10^{34}$.

Acknowledgments

The first author acknowledges Funding by the EU NextGenerationEU through the Recovery and Resilience Plan for Slovakia under the project No. 09I03-03-V02-00036. The first author also acknowledges support from the Grant of Comenius University No. UK/1114/2024. Thirdly, the first author acknowledges support from the grants VEGA Research Grant 1/0437/23 and SK-AT-23-0019 grant.

The second author acknowledges support from the APVV Research Grant APVV-19-0308 and from the VEGA Research Grants 1/0423/20, 1/0727/22 and 1/0437/23.

References

[1] R. Jajcay, G. A. Jones, r-regular families of graph automorphisms, European Journal of Combinatorics 79 (2019) 97–110. doi:https://doi.org/10.1016/j.ejc.2018.12.002.

[2] G. Gauyacq, On quasi-cayley graphs, Discrete Applied Mathematics 77 (1997) 43–58. doi:https://doi.org/10.1016/S0166-218X(97)00098-X.

[3] G. Sabidussi, Vertex-transitive graphs, Monatshefte für Mathematik 68 (1964) 426–438. URL: https://api.semanticscholar.org/CorpusID:120359810.

[4] F. Kerák, r-regular sets of permutations, Bachelor's thesis, Comenius University, Bratislava, 2020.

[5] H. Wilf, generatingfunctionology: Third Edition, CRC Press, 2005. URL: https://books.google.sk/books?id=bVFuBwAAQBAJ.

[6] The GAP Group, GAP – Groups, Algorithms, and Programming, Version 4.12.2, https://www.gap-system.org, 2022.

[7] P. Kollar, R-RegularFamilies, https://github.com/OldAnchovyTopping/R-RegularFamilies, 2024. Accessed: 2024-06-30.