On Cages and Choosing with Symmetries

C. M. De la Cruz[1,2] M. A. Pizaña[1,2]

Submitted: Aug 6, 2025; Accepted: Jan 7, 2026; Published: Mar 27, 2026

Abstract

Given a set S, an integer k and a permutation group G acting on S, we provide a novel algorithm for computing $\binom{S}{k}_G$, which is the set of all k-subsets of S up to symmetries in G. Then we apply our algorithm to compute (d,g)-cages which are d-regular graphs of girth g and minimum order $n = n(d,g)$. Our algorithm allowed us to reproduce most of the computational results on cages and to improve six lower bounds for the order of cages, namely: $n(3,14) \geq 262$, $n(3,15) \geq 388$, $n(3,17) \geq 770$, $n(4,9) \geq 165$, $n(5,7) \geq 110$ and $n(8,5) \geq 69$.

Mathematics Subject Classifications: 05C35, 05C75, 05C85, 05C25

1 Introduction

Detailed, formal definitions follow in the next section.

Take an integer $N > 0$ and the set $\Omega = \{1, 2, \ldots, N\}$. Assume G is a permutation group on Ω and take some G-invariant set $S \subseteq \Omega$ and some integer k with $0 \leq k \leq |S|$. The set of all k-subsets of S is denoted by $\binom{S}{k}$. Now, the action of G on S induces an equivalence relation among the sets in $\binom{S}{k}$, namely, two such k-subsets $X, Y \in \binom{S}{k}$ are equivalent if $Y = X^p$ for some permutation $p \in G$, that is, X and Y are equivalent if the symmetries in G transform one of them into the other. Then the set $\binom{S}{k}_G$ is defined to contain exactly one representative for each of the resulting equivalence classes. Moreover, we ask these representatives to be exactly the smallest k-subset in the class with respect to the lexicographic order. Note that $\binom{S}{k}_G$ is a subset of $\binom{S}{k}$.

It is easy to give an algorithm for computing $\binom{S}{k}$ and to show that it is optimal, however, $\binom{S}{k}$ may be huge as it may grow exponentially with $|S|$ (when $k = |S|/2$ for instance) and hence the algorithm will also have exponential running time. On the other hand, when some permutation group G is involved as explained above, it may be the case that $\binom{S}{k}$ and G are both huge, but $\binom{S}{k}_G$ is small: for example, if $S = \{1, 2, \ldots, n\}$ and G is the symmetric group on S, we have $|G| = n!$ but $\left|\binom{S}{k}_G\right| = 1$. Is it possible to compute

[1]Equal contribution

[2]Universidad Autónoma Metropolitana. clau_mar@ciencias.unam.mx, mpizana@gmail.com

 https://doi.org/10.37236/14471

$\binom{S}{k}_G$ when it is small enough even when both $\binom{S}{k}$ and G are huge? Our algorithm, `Choose(S,k,G)`, does this in practice, at least for our application of interest (computing cages). Ideally, we would like to have a polynomial delay algorithm to compute $\binom{S}{k}_G$, but we do not know yet whether our algorithm has this property.

A (d, g)-graph is a d-regular graph of girth g and a (d, g)-cage is a (d, g)-graph of minimum order $n = n(d, g)$. Cages have been studied extensively [21] since their introduction by Tutte in 1947 [42]. Besides six infinite families of cages there are only eleven parameter pairs where at least one cage with those parameters is known. The known infinite families are cycles ($d = 2$), complete graphs ($g = 3$), complete bipartite graphs ($g = 4$), and also, whenever $d - 1$ is a prime power, $d - 1 = p^\alpha$, there is at least one known (d, g)-cage for $g = 6$, $g = 8$ and $g = 12$. These last three families are incidence graphs of generalized n-gons [43, 41, 5]. The eleven other cases where at least one cage with the given parameters is known, are $n(3, 5) = 10$, $n(3, 7) = 24$, $n(3, 9) = 58$, $n(3, 10) = 70$, $n(3, 11) = 112$, $n(4, 5) = 19$, $n(4, 7) = 67$, $n(5, 5) = 30$, $n(6, 5) = 40$, $n(7, 5) = 50$, $n(7, 6) = 90$. Other than this, there are many known upper and lower bounds for $n(d, g)$. Upper bounds are obtained by providing a (d, g)-graph H (which is not known to be minimal) and hence $n(d, g) \leq |H|$. Lower bounds are obtained either by theoretical results or by exhaustive computer searches. We summarize the results in the literature concerning lower bounds in Table 1 (Section 5).

We compute cages using our algorithm for $\binom{S}{k}_G$. The main idea is to construct graphs incrementally with backtracking, but instead of adding one edge at a time (as is common practice), we compute all possible completions of the neighborhood of a vertex x up to symmetries, taking S as the candidate new neighbors of x, $k = d - \deg(x)$ and G as the stabilizer of x in the automorphism group of the current partial candidate solution (graph) H.

Our techniques allowed us to perform exhaustive computer searches and to improve six lower bounds for the order of cages, namely $n(3, 14) \geq 262$ (previous lower bound 260), $n(3, 15) \geq 388$ (prev. 384), $n(3, 17) \geq 770$ (prev. 768), $n(4, 9) \geq 165$ (prev. 163), $n(5, 7) \geq 110$ (prev. 108) and $n(8, 5) \geq 69$ (prev. 68). We also reproduced many of the known lower bounds and cages, including the eighteen $(3, 9)$-cages, the three $(3, 10)$-cages and the four $(5, 5)$-cages. An exhaustive report of our results can be consulted in Table 2. The lower bounds that we could not reproduce are either theoretical results that are valid for an infinite number of parameters or three computational lower bounds (namely $n(3, 11) = 112$, $n(3, 13) \geq 202$ and $n(4, 7) = 67$) where the corresponding computations required years of CPU time to finish the task (ran in a few weeks on a mixture of machines) [31, 22]. We currently do not have access to such computing power.

All our experiments ran on an Intel Core i7-10700K at 3.8 GHz using a single core and all our programs were written in GAP [23] using YAGS [12] and with the help of `nauty` package [32, 33]. We make heavy use of several techniques, including: backtracking, dynamic programming, lazy evaluation, path compression, case and conflict reduction using symmetries, conflict selection, speculative exploration and the standard orbit-stabilizer algorithm. Our main contribution for computing cages comes from the use of our algorithm for generating $\binom{S}{k}_G$.

The code and explanations in this paper are optimized for clarity. However, our actual code is optimized for efficiency and hence deviates from the one explained here in several ways. For full details, see the actual code [15], but we do explain the main differences at the end of each section. An early report of this work was submitted as an extended abstract to LAGOS 2025 [16].

2 Preliminaries and terminology

Let $N \in \mathbb{N}$, $\Omega = \{1, 2, \ldots, N\}$, G a permutation group on Ω and $S \subseteq \Omega$ a G-invariant subset. We shall use these symbols consistently across the paper.

2.1 Lists and sets. *Lists* are ordered tuples and we write them using square brackets, like $L = [x_1, x_2, \ldots, x_n]$ and $M = [y_1, y_2, \ldots, y_m]$. We say that L is a *prefix* of M, denoted as $L \preccurlyeq M$, if and only if $n \leq m$ and $x_s = y_s$ for all $s \leq n$; it is a *proper prefix*, denoted as $L \prec M$, if we also have $n < m$. For a list $L = [x_1, x_2, \ldots, x_n]$ and $t \leq n$, we define $L[\leq t] = [x_1, x_2, \ldots, x_t]$. Given a set X, we denote the set of all lists with elements in X as X^*.

We say that a set expression like $\{x_1, x_2, \ldots, x_n\}$ is in *sorted presentation* if $x_i < x_{i+1}$ for all i. Hence these two sets are the same: $\{2, 5, 7, 9\} = \{7, 5, 2, 9\}$, but the first expression is in sorted presentation and the second is not. Sets of integers can be considered as lists using their sorted presentations, hence we may also say that a set X is a *prefix* (*proper prefix*) of another set Y, which we also denote as $X \preccurlyeq Y$ ($X \prec Y$). Note that $X \preccurlyeq Y$ implies $X \subseteq Y$. Given a set X and a number z, we define $X_{<z} = \{x \in X : x < z\}$. Clearly $X_{<z} \preccurlyeq X$. As usual, $\max X$ is the maximum element of X. For a set X and an integer k, we denote the set of all k-subsets of X by $\binom{X}{k} = \{Y \subseteq X : |Y| = k\}$. We shall need the following strict partial order among subsets of Ω:

Definition 1. Let $X, Y \subseteq \Omega$, with $X = \{x_1, x_2, \ldots, x_n\}$, $Y = \{y_1, y_2, \ldots, y_m\}$ in sorted presentation. We say that Y *precedes* X, denoted as $Y < X$, if and only if $\exists\, s \leq n, m$ with $y_s < x_s$ and $y_j = x_j$ for all $j < s$.

Note that this strict partial order is very similar to the lexicographic order except that here whenever Y is a prefix of X, $Y \preccurlyeq X$, we have $Y \not< X$ and $X \not< Y$. However, our strict partial order does coincide with the lexicographic order whenever X and Y have the same cardinality.

2.2 Graphs and cages. Our graphs $H = (V, E)$ are finite and simple. We use standard terminology [8, 24] for the *vertex set* $V(H)$, *edge set* $E(H)$, *degree* of a vertex $\deg(x) = \deg_H(x)$, *maximum degree* $\Delta(H)$, and *distance* between vertices $\mathrm{dist}(x, y) = \mathrm{dist}_H(x, y)$. We denote the *disjoint union of graphs* as $H_1 + H_2 = (V(H_1) \cup V(H_2), E(H_1) \cup E(H_2))$. We also find it convenient to denote the graph resulting from H by adding an edge (or a set of edges) by $H + e$ (or $H + \{e_1, e_2, \ldots, e_r\}$).

A graph H is *d-regular* if $\deg_H(x) = d$ for all $x \in V(H)$. The *girth* of a graph, $\mathrm{Girth}(H)$, is the length of the shortest cycle in H. Given $d \geq 2$ and $g \geq 3$ a *(d, g)-graph* is

a d-regular graph of girth g. A (d,g)-*cage* is a (d,g)-graph of minimum order. We denote by $n(d,g)$ the order of a (d,g)-cage. Cages for $d=2$ (cycles), $g=3$ (complete graphs) and $g=4$ (complete bipartite graphs) are generally considered trivial and hence we shall focus only on (d,g)-cages for $d\geq 3$ and $g\geq 5$.

A fundamental lower bound for the order of cages is the celebrated *Moore bound* (as attributed by Hoffman and Singleton in [25], although the version for even girth seems to be due to Erdős and Sachs [17]): Every (d,g)-graph must contain a *Moore tree*, which is a d-regular (save for leafs) tree of diameter $g-1$ and maximum order. The order of such a tree is the Moore bound, $M(d,g)$, and its numerical value is:

$$M(d,g)=\begin{cases} 1+d\cdot\sum_{k=0}^{(g-3)/2}(d-1)^k &= \frac{d(d-1)^{(g-1)/2}-2}{d-2} \quad \text{when } g \text{ is odd.}\\ \\ 2\cdot\sum_{k=0}^{(g-2)/2}(d-1)^k &= \frac{2(d-1)^{g/2}-2}{d-2} \quad \text{when } g \text{ is even.}\end{cases}$$

It follows that $n(d,g)\geq M(d,g)$. We further abbreviate $M(d,g)$ simply as mb when d and g are clear from the context. The number $e(d,g)=n(d,g)-mb$ is called the *excess* and the graphs obtained by removing a Moore tree from a cage are called *excess graphs*.

A trivial but fundamental observation is that there can not exist (d,g)-graphs of odd order n when d is odd (see McGee [30]), due to Euler's Handshaking Lemma [19].

2.3 Permutations, groups and actions. We shall only consider permutation groups. A *permutation* $p:\Omega\to\Omega$ is a bijective function on some domain Ω. The trivial permutation is denoted by 1. The *symmetric group* on Ω, $\mathcal{S}_\Omega$, is the collection of all such permutations with (right) composition as group operation. We use exponential notation for the action of a permutation $p\in\mathcal{S}_\Omega$ on an element $x\in\Omega$, hence $x^p=p(x)$ and also $x^{pq}=(x^p)^q=q(p(x))$. If $L=[x_1,x_2,\ldots,x_n]\in\Omega^*$ is a list and $X=\{x_1,x_2,\ldots,x_n\}\subseteq\Omega$ is a set, permutations in $\mathcal{S}_\Omega$ act naturally on them: $L^p=[x_1^p,x_2^p,\ldots,x_n^p]$, $X^p=\{x_1^p,x_2^p,\ldots,x_n^p\}$. Note that the latter expression, in general, is not in sorted presentation.

Our permutation groups, G, will always be subgroups of $\mathcal{S}_\Omega$, (denoted as $G\leq\mathcal{S}_\Omega$). Our set $S\subseteq\Omega$ is required to be G-*invariant*, that is, such that $S^p=S$ for all $p\in G$. Obviously, $L\in S^*$ implies $L^p\in S^*$ and $X\subseteq S$ implies $X^p\subseteq S$. The *orbits* of elements, lists and sets are defined as $x^G=\{x^p:p\in G\}$, $L^G=\{L^p:p\in G\}$, $X^G=\{X^p:p\in G\}$. Since S is G-invariant, we have $S^G=\{S\}$.

The *stabilizer* of x in G is the subgroup generated by all the permutations of G that fix x, that is $G_x=\langle\{p\in G:x^p=x\}\rangle$. *Iterated stabilizers* are denoted by simply concatenating the subscripts: $G_{x_1x_2}=(G_{x_1})_{x_2}$, $G_{x_1x_2x_3}=((G_{x_1})_{x_2})_{x_3}$ and so on. Given two elements x,y, a *transporter* from x to y is a permutation $p\in G$ such that $y=x^p$. The set of all such transporters is denoted by $\mathrm{Tr}(G,x,y)$. Note that $\mathrm{Tr}(G,x,y)$ is empty whenever $x^G\neq y^G$. If $p\in\mathrm{Tr}(G,x,y)$, it should be clear that $\mathrm{Tr}(G,x,y)=pG_y=G_xp$ (these last two expressions are *left and right cosets* of G_y and G_x, respectively, in G). Stabilizers and transporters of lists and sets in G are defined in the analogous way, in particular $\mathrm{Tr}(G,M,L)=\{p\in G:L=M^p\}$.

Consider $\binom{S}{k}$, as in Section 2.1, and note that it is G-invariant, that is, for every $X\in\binom{S}{k}$, and $p\in G$ we have that $X^p\in\binom{S}{k}$. Hence, the action of G on $\binom{S}{k}$ induces

an equivalence relation on $\binom{S}{k}$: $X \sim Y$ if and only if $X^G = Y^G$. This in turn, induces a partition of $\binom{S}{k}$ where the parts are precisely the orbits of the form X^G. We are interested in computing one representative for each equivalence class of this partition. Moreover, we ask this representative to be minimum in its class with respect to the strict partial order in Definition 1. Hence we define:

$$\binom{S}{k}_G = \left\{X \in \binom{S}{k} : \forall Y,\ Y^G = X^G \text{ and } Y \neq X \Rightarrow X < Y\right\}.$$

We shall present an algorithm, `Choose(S,k,G)`, for computing this.

GAP uses arrays to represent permutations on Ω: $p = [1^p, 2^p, \ldots, N^p]$, and hence, actions of permutations on elements, x^p, can be obtained in $\Theta(1)$ time, but multiplications and inverses of permutations are computed $\Theta(N)$ time. Groups are represented as a list of generators (permutations).

2.4 Orbit-stabilizer algorithm. When computing a transporter or a stabilizer, GAP uses the Schreier-Sims algorithm [40] to produce a full stabilizer chain, which is much more than what we need in our algorithms. Therefore, for efficiency reasons, we had to reimplement those two procedures following the guidelines of Seress [39] and Hulpke [26] as described in this section.

Given a group $G \leq \mathcal{S}_\Omega$ and an element $x \in \Omega$, a *standard orbit-stabilizer algorithm* [39] produces a rooted tree containing all the elements of the orbit x^G of x. The tree has x at the root and for each node $y \in x^G$, and for each generator p of G, y^p is a child of y, unless y^p is already present at some other place in the tree. These trees are usually represented by a pair of arrays `parent` and `ptransporter` (parent transporter) in such a way that `parent[y^p] = y` and `ptransporter[y^p]= p`. The orbit of x is then represented by this tree. Also, note that all the orbits can be represented by such trees using a single pair of such arrays of length $N = |\Omega|$. The trees produced by the standard orbit-stabilizer algorithm, can be used to calculate orbits, transporters, stabilizers and more:

During the construction of each of these trees, the orbit of x (represented as a set) is usually computed and stored. For efficiency reasons, and using two additional arrays, it is also convenient to store the root and root transporter of each node such that `root[y] = x` and `rtransporter[y] = p` whenever $y = x^p \in x^G$ (here p is the product of the parent transporters of y and its ancestors). Hence, we can easily check whether two elements y_1, y_2 belong to the same orbit by simply checking the condition `root[y1] = root[y2]`. Also, we can obtain a transporter from y_1 to y_2 (when it exists) as

```
Transporter(G,y1,y2) = rtransporter[y1]^(-1)*rtransporter[y2].
```

We can also use these trees to generate the stabilizer G_x, since, thanks to Schreier's Lemma [40, 39], it is generated by all the elements of the form

```
rtransporter[y]*p*rtransporter[y^p]^(-1)
```

where y iterates over all the leafs of the tree of x and p iterates over all the generators of G. However, to avoid unnecessary computations, we only compute roots, root transporters and stabilizers when actually needed.

Recall that $N = |\Omega|$, let N_G be number of generators of G, it should be clear that the time required to construct the tree of x is $\Theta(|x^G| \cdot N_G)$ and constructing all the trees for every orbit representative takes a total of $\Theta(N \cdot N_G)$ time, while computing all the roots requires $O(|x^G|)$ time per tree, $O(N)$ time for all trees. With all the trees at hand, and the roots already obtained, we can decide whether two elements $y_1, y_2 \in \Omega$ belong to the same orbit in $\Theta(1)$ time. Computing individual root transporters may take anywhere from $\Omega(1)$ to $O(|x^G| \cdot N)$ time (since multiplication of permutations requires $\Theta(N)$ time), but computing all of them for a given tree, requires $O(|x^G| \cdot N)$ time if we already have the tree of x. Once you have the root transporters, any transporter can be produced in $O(N)$ time (the time required for inverting one permutation and multiplying it by another). Producing (the list of generators of) a stabilizer takes $O(|x^G| \cdot N_G \cdot N)$ additional time if we already have the tree and the root transporters. All of these are stored when computed, and hence, any of this takes $\Theta(1)$ additional time when required again.

Since the time complexity of group algorithms depend on the number of generators N_G, it is common practice to try to reduce the number of generators when computing a new group (a stabilizer for instance) by discarding any new generator whenever it already belongs to the group generated by the previous generators. However doing so requires computing a full stabilizer chain, and according to our computational experiments, it takes too much additional time for our problems of interest. Instead we simply discard any new generator whenever it is identical to a generator already produced. This reduction step takes $O(|x^G| \cdot N_G \cdot N \cdot \log(N_{G_x}))$ additional time for computing stabilizers (where N_{G_x} is the number of generators of the resulting stabilizer G_x).

The Schreier-Sims algorithm uses the ideas of the standard orbit-stabilizer algorithm recursively to produce a stabilizer chain mentioned above. The stabilizer chain is very useful since it can be used to obtain the order of a group, determine whether a given permutation belongs to a given group, decompose a permutation as a product of generators, produce different stabilizer chains, and more. But it represents much more work than we need in our algorithm.

2.5 Finding transporters for lists. We present here this standard algorithm (used in GAP, for instance), since the ideas involved are heavily used in our algorithms in the next section. Given two lists $M, L \in S^*$ of the same length, we want to find a transporter $p \in \mathrm{Tr}(G, M, L)$ (if there is one). Clearly, iterating p over the elements of G is not feasible in general as G may be huge. On the other hand, the standard orbit-stabilizer algorithm would generate the orbit of M until L is found or the orbit of M is completed. This may be feasible if the orbit of M is small (see section 3.4), but not in general, since the orbit of M may also be huge (as large as $\binom{N}{k}$ with $k = |M|$). However, the following theorem tells us that we may determine the existence of p and its value, with a greedy algorithm.

Theorem 2. *Let $G \leq \mathcal{S}_\Omega$ and let $L = [x_1, x_2, \ldots, x_n]$, $M = [y_1, y_2, \ldots, y_n]$ be lists of the same length in S^*. Then $\exists\, p \in G$ such that $L = M^p$ if and only if all the following conditions hold.*

1. *$\exists\, p_1 \in G$ with $x_1 = y_1^{p_1}$.*

2. $\exists\, p_2 \in G_{x_1}$ *with* $x_2 = y_2^{p_1 p_2}$.

3. $\exists\, p_3 \in G_{x_1 x_2}$ *with* $x_3 = y_3^{p_1 p_2 p_3}$.

$\vdots$

n. $\exists\, p_n \in G_{x_1 x_2 \cdots x_{n-1}}$ *with* $x_n = y_n^{p_1 p_2 \cdots p_n}$.

Moreover, if $p_1, p_2, \ldots, p_t$ *satisfy conditions* (1)-(t) *for some* t *with* $0 \le t < n$ *and there is no* p_{t+1} *satisfying (t+1), then* $\nexists\, p \in G$ *such that* $L = M^p$.

Proof. Assume $L = M^p$. We can set $p_1 = p$ and $p_j = 1$ for all $j \ge 2$ and then $p_1, p_2, \ldots, p_n$ satisfy conditions (1)-(n). Conversely, assume $p_1, p_2, \ldots, p_n$ satisfy conditions (1)-(n). Now take $p = p_1 p_2 \cdots p_n$ and note that $L = M^p$.

Finally, assume $p_1, p_2, \ldots, p_t$ satisfy conditions (1)-(t) for some t with $0 \le t < n$ and there is no p_{t+1} satisfying $(t+1)$. By way of contradiction, assume further that $L = M^p$ for some $p \in G$. By the previous paragraph, there are $p'_1, p'_2, \ldots, p'_n$ satisfying conditions (1)-(n). Now take $q = p_1 p_2 \cdots p_t$ and $q' = p'_1 p'_2 \cdots p'_t$. Note that for all $j \le t$, we have $x_j = y_j^q$ and $x_j = y_j^{q'}$. It follows that $x_j^{q^{-1}q'} = x_j$ for all $j \le t$ and hence that $q^{-1}q' \in G_{x_1 x_2 \cdots x_t}$. Then take $p_{t+1} = q^{-1} q' p'_{t+1}$ and note that $p_1, p_2, \ldots, p_{t+1}$ satisfy $(t+1)$. A contradiction. □

Hence we can compute `Transporter(G,M,L)` as follows.

```
Transporter(G,M,L){
   n = Length(L); G1 = G;
   p = TrivialPermutation();
   for(i = 1; i <= n; i++){
      p1 = Transporter(G1,M[i]^p,L[i]);
      if(p1 == fail){return fail;}
      p = p*p1;
      G1 = Stabilizer(G1, L[i]);
   }
   return p;
}
```

Note that the previous algorithm allows us to decide the condition

```
(∃ p ∈ G with L= M^p).
```

2.6 Some implementation details. Some products of permutations, like the ones used to compute stabilizers, are maintained as a list of permutations until they are actually needed.

3 Choosing with symmetries

3.1 The general idea. This section explains how we compute $\binom{S}{k}_G$ = `Choose(S,k,G)`. The general idea is simple: Use backtracking to compute $\binom{S}{k}$ in lexicographic order and pruning whenever the partial solution considered, X, has been previously considered up to symmetries in G. Recall the strict partial order $X < Y$ in Definition 1, with this order, the pruning condition is simply:

```
PreviouslyConsidered(G,X){return (∃ p ∈ G with X^p < X);}
```

Obviously, a direct (naive) implementation of the previous code is not feasible in general, since the group G may be huge and hence, iterating p over the elements of G is not feasible. Instead, we use the theorems in this section to implement that function much more efficiently.

3.2 Finding covers for sets. Given a group $G \leq \mathcal{S}_\Omega$ and two sets $X, Y \subseteq \Omega$, we shall need to decide the condition `(∃ p ∈ G with Y ⊆ X^p)`. When that condition holds, we say that X *covers* Y. We already know that it is not feasible to iterate over the elements p of G. It is also not feasible to consider Y as a list L and then generate all the lists $M \in X^*$ with $|M| = |L|$ (which are, in general, too many) to finally use the algorithm in Subsection 2.5 to determine whether `(∃ p ∈ G with L=M^p)`.

Instead, we may indeed consider Y as a list L using its sorted presentation, but then use backtracking to generate the lists $M \in X^*$ with length at most $|L|$, but pruning whenever the partial solution considered, M, can not be extended. That is, consider the list $L = [y_1, y_2, \ldots, y_m]$ and assume $M = [x_{w_1}, x_{w_2}, \ldots, x_{w_t}]$ (not sorted in general) is the partial solution being considered, which means that `(∃ p ∈ G with L[≤t]=M^p)`. Then, when $t < m$, we try to extend M with an additional $x_{w_{t+1}}$, that is $M = [x_{w_1}, x_{w_2}, \ldots, x_{w_{t+1}}]$, but we find out that now `(∄ p ∈ G with L[≤t+1]= M^p)`, which can be computed as in Subsection 2.5. Then we prune the backtracking search tree here and try the next available $x_{w_{t+1}}$; if there is no such $x_{w_{t+1}}$ available, then we backtrack to $M = [x_{w_1}, x_{w_2}, \ldots, x_{w_{t-1}}]$, try the next available x_{w_t} and continue the search there.

Even better, since the partial solutions M are generated incrementally, we can integrate the pruning procedure into this backtracking procedure in such a way that the pruning procedure does not need to start from scratch each time, but instead, we maintain a current list of permutations $p_1, p_2, \ldots, p_t$ (as in Theorem 2) and check if the current list of permutations can be extended with an additional permutation p_{t+1} whenever M has just increased in length. For future reference, we shall denote the just described procedure as

`Cover(G,Y,X)`,

which is semantically equivalent to

`(∃ p ∈ G with Y ⊆ X^p)`.

3.3 Computing `PreviouslyConsidered(G,X)` and `Choose(S,k,G)`. Recall from Subsection 3.1 that the procedure `PreviouslyConsidered(G,X)` described there, is our main pruning condition for computing $\binom{S}{k}_G$ and that it is semantically equivalent to `(`$\exists p \in G$ `with` $X^p < X$`)`. We still need the following additional theorems to describe our implementation of it. We begin by characterizing the precedence relation:

Theorem 3. *Let $X, Y \subseteq \Omega$ and $x_n = \max X$. Then the following statements are equivalent:*

1. $Y < X$.
2. $\exists\, z \in Y \setminus X$, $z < x_n$ *with* $X_{<z} = Y_{<z}$.
3. $\exists\, z \in Y \setminus X$, $z < x_n$ *with* $X_{<z} \subseteq Y$.

Proof. (1 $\Leftrightarrow$ 2) Just take $z = y_s$ with y_s as in Definition 1. (2 $\Leftrightarrow$ 3) Note first that $X_{<z} \subseteq Y$ if and only if $X_{<z} \subseteq Y_{<z}$. If this last inclusion is proper, then, there is a $z' \in Y_{<z} \setminus X_{<z}$, smaller that z, with all the properties of z in (3). Therefore, the minimum z satisfying (3) must also satisfy (2). □

In particular, when $Y = X^p$, the following characterizations of $X^p < X$ are useful:

Theorem 4. *Let $X \subseteq S$, $p \in G$ and $x_n = \max X$. Then, the following conditions are equivalent.*

1. $X^p < X$
2. $\exists\, z \in X^p \setminus X$, $z < x_n$, *with* $X_{<z} \subseteq X^p$.
3. $\exists\, z \in X^p \setminus X$, *with* $X_{<z} \subseteq X^p$.
4. $\exists\, z \in S_{<x_n} \setminus X$, *with* $X_{<z} \cup \{z\} \subseteq X^p$.

Proof. Simply by setting $Y = X^p$, we get (1) $\Leftrightarrow$ (2) by Theorem 3. Clearly (2) $\Rightarrow$ (3). Now, assume (3). If $x_n < z$ then we get $X \cup \{z\} = X_{<z} \cup \{z\} \subseteq X^p$, but $|X \cup \{z\}| > |X^p|$, a contradiction. It follows that $z < x_n$ and hence that (3) $\Rightarrow$ (2). Finally (2) $\Leftrightarrow$ (4) is straight forward considering that S is G-invariant and hence $X^p \subseteq S$. □

Since we will generate $\binom{S}{k}_G$ using backtracking, it is good to prune as early as possible. In particular, the following theorem says that, if at some given time, we are considering Y as a partial solution, in our way to find a full solution X in $\binom{S}{k}_G$ (and hence Y is a prefix of X), but we find out that Y has been previously considered (i.e. $Y^p < Y$ for some p in G), then any possible full solution X (satisfying $Y \preccurlyeq X$) has also been previously considered ($X^p < X$ for some $p \in G$). This allows us to prune the whole search subtree at Y.

Theorem 5. *$Y \preccurlyeq X$ and $Y^p < Y \Rightarrow X^p < X$.*

Proof. Let $x_n = \max X$ and $y_m = \max Y$. By Theorem 4, we have some $z \in Y^p \setminus Y$, $z < y_m$, such that $Y_{<z} \subseteq Y^p$. Since $Y \subseteq X$, we have $Y^p \subseteq X^p$. Since $Y \preccurlyeq X$ and $z < y_m$, if we had $z \in X$, then we get $z \in Y$ a contradiction. It follows that $z \in X^p \setminus X$. Clearly $z < x_n$. Finally, observe that $X_{<z} = Y_{<z} \subseteq Y^p \subseteq X^p$. By Theorem 4, it follows that $X^p < X$. □

Now assume that $X = \{x_1, x_2, \ldots, x_n\}$ is in sorted presentation. Assume also that X is a partial solution currently being considered in our generation of $\binom{S}{k}_G$. Since we are generating them using backtracking, no proper prefix of X, $Y \prec X$, satisfy $Y^p < Y$ for some p in G (since otherwise, by the previous theorem, the subtree at Y should have been pruned earlier). Then, the greatest element of X, $x_n = \max X$, is the only element in X which is not present in any proper prefix of X. It follows that x_n must actively participate in fulfilling the condition $X^p < X$ (when it holds). Then, with the characterization (and notation) given in Theorem 4, x_n must participate either by covering z (i.e. $z = x_n^p$ for some p) or by covering some $x_s < z$ (i.e. $x_s = x_n^p$ for some $x_s \in X_{<z}$). The following theorem makes these ideas more precise.

Theorem 6. *Let $X \subseteq S$, $x_n = \max X$ and $p \in G$. Assume $X^p < X$. Let z be as in Theorem 4(4). Furthermore, assume that for every proper prefix $Y \prec X$ we have that $Y^p \not< Y$. Then one of the following conditions hold:*

1. $x_n^p = z$.
2. $x_n^p \in X_{<z}$.

Proof. The theorem is trivially true, when $n = 1$. Hence assume $n \geq 2$. By Theorem 4 we have $z \in S_{<x_n} \setminus X$, with $X_{<z} \cup \{z\} \subseteq X^p$. Let $Y = X_{<x_n} = X \setminus \{x_n\}$, note that $x_{n-1} = \max Y$. By hypothesis, $Y^p \not< Y$. By way of contradiction, assume further that $x_n^p \neq z$ and $x_n^p \notin X_{<z}$.

We know that $X_{<z} \cup \{z\} \subseteq X^p$. If we had $z > x_{n-1}$, then $|X_{<z} \cup \{z\}| = |X| = |X^p|$, which implies $x_n^p \in X_{<z}$ or $x_n^p = z$ a contradiction. It follows that $z < x_{n-1}$ and hence $z \in S_{<x_{n-1}} \setminus X = S_{<x_{n-1}} \setminus Y$. Finally $Y_{<z} \cup \{z\} \subseteq X_{<z} \cup \{z\} \subseteq X^p \setminus \{x_n^p\} = Y^p$. By Theorem 4 we get $Y^p < Y$ a contradiction. □

Using Theorems 4, 5 and 6 we may code `PreviouslyConsidered(G,X)` as follows.

```
PreviouslyConsidered(G,X){
   if( IsTrivial(G) ){return false;}
   xn= max X; Z= [1,2,\dots,xn]\X;
   for(z in Z){
      p=Transporter(G,xn,z); if(p==fail){continue;}
      G1=Stabilizer(G,z); Xz = { x in X : x<z };
      if( Cover(G1, Xz, X^p) ){ return true; }
   }
   zmax = max(Z);
   for(xs in X with xs< zmax){
      p=Transporter(G,xn,xs);if(p==fail){continue;}
      G1=Stabilizer(G,xs);
      for(z in Z with z > xs){
```

```
            Xz = Union( {z}, { x in X: x<z, x<>xs });
            if( Cover(G1, Xz, X^p) ){return true;}
        }
    }
    return false;
}
```

Using this pruning condition, we implement the procedure `Choose(S,k,G)` to compute $\binom{S}{k}_G$ as described at the beginning of this section. We also implement the procedure `ChooseNext(S,k,G,L)`, which gives one solution at a time. Here `L` is stores the state of the computation so that we can continue it and find the next solution at a later time. Initially, its value is `L=[]`, the empty list.

3.4 Some implementation details. The procedure `PreviouslyConsidered` has the additional parameter `S`, as in `PreviouslyConsidered(G,S,X)`. In this way, we may compute `Z=S\X;` (third line of the previous code) which may be substantially smaller than `Z=[1,2,...,xn]\X;`. Also, the line `Xz = Union({z},...)` is actually a concatenation of lists: in this way, `Cover(G1, Xz, X^p)` (which treats `Xz` as a list) tries to cover `z` first and fails immediately if that is not possible.

Let $r = |X|$. When $|\binom{S}{r}| < 2^{16}$, a standard orbit-stabilizer algorithm works faster than our procedures, hence we do use this alternative when this condition is met. The threshold value 2^{16} was determined experimentally.

The procedures `Choose` and `ChooseNext` have an additional parameter `Check`, as in `ChooseNext(S,k,G,Check)` and `ChooseNext(S,k,G,Check,L)`. `Check(X)` is a user-provided procedure to be used as an additional pruning condition during backtracking. In such a way, the user can adapt our code to meet additional requirements for the particular problem being solved. We shall use it in the next section for pruning the search tree when the partial solution considered (a graph) fails to meet the girth condition.

4 Searching for cages

We use a backtracking algorithm to do exhaustive searches for (d, g)-graphs and (d, g)-cages. For this, we maintain a list (stack), `LH`, of partial candidate solutions: *Partial candidate solutions* are graphs, H, with $\Delta(H) \leq d$ and $\mathrm{Girth}(H) \geq g$. A *conflict* is a vertex $x \in H$ with $\deg(x) < d$. We say that a cycle of length less than g is a *short cycle*. A *solution, sol, for a conflict* x is a set of vertices $sol \subseteq V(H)$ satisfying:

1. $|sol| = d - \deg(x)$.
2. For all $y \in sol$: $y \neq x$, $\deg(y) < d$ and $xy \notin E(H)$.
3. $H_1 = H + \{xy : y \in sol\}$ has no short cycles.

In general, a conflict may have many solutions. Each solution for a conflict gives rise to a new graph H_1 as above. The vertex x is no longer a conflict in H_1 and hence H_1 has fewer conflicts than H.

Our main contribution here is to compute all the solutions for a conflict up to symmetries of H using our procedure `ChooseNext(S,k,G,Check,L)` from the previous section (more on this below). The general idea of the algorithm can be expressed as follows:

```
GenAll(d,g,n){
   LH=Initialize(d,g,n); LSols=[];
   while( LH <> [] ){
      H=Pop(LH);
      x=SelectConflict(H);
      Lnew=SolveConflict(H,x);
      for( H1 in Lnew ){
         if( IsSolution(H1) ){
            Add(LSols,H1);
         }else{
            Push(LH,H1);
        }
      }
   }
   return LSols;
}
```

We explain the subroutines in the previous code as follows:

`Initialize(d,g,n)`: The initial stack `LH` may be set to be a list containing only the discrete graph on n vertices (n isolated vertices) and the algorithm works well: it decides to start by constructing the Moore tree for the (d, g)-graph and does so very quickly. However we do construct the Moore tree, `T`, first so we take all the excess vertices `Ex` (those not in `T`) and compute all the possible *excess graphs* `E1, E2, ..., Er` with vertex set `Ex` that satisfy Δ`(Ei)` $\leq$ `d` and `Girth(Ei)` $\geq$ `g`. We do this using Brendan McKay's programs `geng` and `pickg` included with `nauty` [32, 33]. Finally we set the initial value of `LH` as `LH=[T+E1,T+E2,..., T+Er]`, where '`+`' stands for the disjoint union of graphs. This initialization provides an important time improvement when the excess is not very small. In particular, note that pairs of excess vertices need not be considered as possible additional edges for any partial candidate solution H since any such additional edge added to some excess graph simply produces another of the excess graphs or breaks the degree or the girth condition.

`SelectConflict(H)`: Among the several conflicts `x` in `H`, we select one with the minimum number of solutions. This has a profound effect on the total number of cases (partial candidate solutions) considered by the algorithm and hence on the total running time.

For this, we compute automorphism group `G` of `H`, using `nauty` [32, 33]. We provide `nauty` with the partition that distinguishes excess vertices from Moore tree vertices, since those should not be mixed (given that additional edges between excess vertices will not be considered). Then we generate the list of conflicts, `confs`, of `H` up to symmetries in `G`. When trying to solve a conflict $x \in$ `confs`, the relevant group is $G_x =$ `Stabilizer`(G, x) because we are looking for neighbors of x itself and not neighbors of some other x^p equivalent

to x under G. Thus, we could solve each conflict $x \in$ `confs` using `Choose(S,k,G`$_{\texttt{x}}$`,Check)` but this may be slow, since some conflicts may have a huge number of solutions. Instead we use a transversal approach: we find one solution at a time for each conflict using `ChooseNext(S,k,G`$_{\texttt{x}}$`,Check,L)` and, as soon as one conflict fails to have an additional solution, it certainly has the minimum number of solutions among all conflicts. Recall that in `ChooseNext(S,k,G`$_{\texttt{x}}$`,Check,L)`, `L` simply maintains the state of the computation so we can compute the next solution at a later time. Hence, for a given conflict $x \in$ `confs`, we take `G`$_{\texttt{x}}$ as described above, $k = d - \deg(x)$ and

$$\texttt{S} = \{y \in V(H) : \deg(y) < d, \operatorname{dist}(x,y) \geq g-1 \text{ and } \{x,y\} \notin \texttt{Ex}\}.$$

Finally we use `Check(X)` to test whether the edges considered from a given solution, $\{xy : y \in sol\}$, form short cycles when added to H, so we can prune the search in `ChooseNext(S,k,G`$_{\texttt{x}}$`,Check,L)` when this happens.

`SolveConflict(H,x)`: The selected conflict `x` has a (possibly empty) list of solutions `sols=[sol1, sol2, ..., solr]`, which was indeed already computed in the previous step (and stored for later use). Now, for each solution `solj`, we compute the new graph `Hj` = `H` + {`xy` : `y` $\in$ `solj`} and return the list of new partial candidate solution graphs `Lnew` `=[H1,H2,...,Hr]`.

`IsSolution(H)`: Simply checks whether `H` is already free of conflicts, and hence, whether it is already a (d,g)-graph.

4.1 Correctness testing. Many standard techniques were extensively used during the software development process in order to promote correctness of the implementation, including: unit testing, regression testing, independent implementation of critical modules, differential testing, redundant testing (via assertions) and exploratory testing.

For instance, every version of `ChooseNext(S,k,G,Check,L)` was verified to produce identical results as the previous version for our test suite. The test suite consisted of all calls to `ChooseNext(S,k,G,Check,L)` required by the cage searches that could be completed within one hour for a given version. These versions include implementations derived from algorithms by Pech and Reichard [37] and Linton [29], as well as a very straightforward but considerably slower initial version. In addition, the total number of partial candidate solutions `H` considered by `GenAll(d,g,n)` was verified to match the corresponding total for the previous versions whenever applicable. Moreover, as noted before, our experimental results on cages match the known results in the literature.

4.2 Some implementation details. The pruning condition `Check(X)` does not actually compute the girth of `Hj`. Doing so would require an $O(n^3)$ call to the girth procedure for each graph `Hj` in `Lnew`, as well as for each candidate graph that was considered for `Lnew` but discarded for failing the girth condition.

Instead, for each vertex y in `H`, we maintain two lists: the vertices at distance at most $\lfloor \frac{g-3}{2} \rfloor$ and the vertices at distance at most $\lceil \frac{g-3}{2} \rceil$. Using these lists, `Check(X)` can

determine whether two additional edges xy_1 and xy_2 from a solution `solj` form a short cycle by computing a set intersection and checking whether it is non-empty. Computing these lists takes $O(n^2)$ time per vertex ($O(n^3)$ for all vertices), but they are computed only for conflict vertices (those with $\deg(x) < d$) and only once per graph `H`. After the lists have been computed, each girth verification requires only $O(n)$ time, which is the cost of computing the intersection. This is done without constructing the corresponding graph `Hj`.

We proceed similarly for the condition $\text{dist}(x, y) \geq g - 1$ required in the definition of S above. All lists are computed and stored on demand, and are recomputed from scratch whenever a new graph `H` is considered. The lists are implemented as *blists* (binary lists), allowing fast set operations.

This improvement alone made our algorithms run up to eight times faster in some test cases. We are grateful to Geoffrey Exoo for suggesting this idea.

5 New lower bounds for cages

A number of theoretical and computational results concern lower bounds of cages. For the reader's convenience we summarize them in Table 1.

Our algorithm ran on a single processor, Intel Core i7-10700K at 3.8 GHz always using a single core. We did run up to 8 processes at a time. The maximum time allotted per case was three months, but many processes were aborted after two days when no significant progress was achieved in that time.

Table 2 shows our experimental results comprehensively. There, a check mark (✓) indicates an improved lower bound, a cross mark (✗) indicates we were unable to reproduce a previously known lower bound. A red hyphen (-) indicates that our algorithm took too long even to deal with the Moore bound. We improved six lower bounds: $n(3, 14) \geq 262$, $n(3, 15) \geq 388$, $n(3, 17) \geq 770$, $n(4, 9) \geq 165$, $n(5, 7) \geq 110$ and $n(8, 5) \geq 69$. The results that we could not reproduce are either theoretical results that are valid for an infinite number of parameters or three previously known computational results (namely $n(3, 11) = 112$, $n(3, 13) \geq 202$ and $n(4, 7) = 67$) where the authors reported years of CPU time (ran in a few weeks on a mixture of machines) to obtain the lower bound. We currently do not have access to such computing power. A reimplementation in C language and the use of more computing power could likely yield some additional lower bound improvements.

In all cases, we started with the Moore bound (even if better bounds were known) and try higher and higher bounds when feasible, of course skipping odd orders n when d was odd. For instance, our bound $n(4, 9) \geq 165$ means that all values from 161 to 164 were tried and our algorithm found no $(4, 9)$-cages of those orders, and hence the new lower bound is 165. Whenever our lower bound matched an upper bound (and hence a cage of that order exists) for instance in $n(3, 9) = 58$, we were able to finish the search for the case and find all the known cages for those parameters (all the eighteen $(3, 9)$-cages on 58 vertices in this case).

id	Result	Hypotheses	Ref.	Year
	$n \geq mb$	None (Moore bound)	[25, 17]	1960, 1963
	$n \equiv 0 \mod 2$	$d \equiv 1 \mod 2$ (Handshaking Lemma)	[19, 30]	1741, 1960
	$n = mb$	$g = 3, 4$ or $d = 2$	[42, 17]	1947, 1963
a	$n = mb$	$g = 5$ and $d = 2, 3, 7$	[25]	1960
b	$n = mb$	$g = 6, 8, 12$ and $d = p^\alpha + 1$	[43, 41, 5]	1906, 1966
c	$n \neq mb$	$g \neq 3, 4, 5, 6, 8, 12$ and $d \neq 2$	[41, 3, 14]	1966, 1973
d	$n \neq mb$	$g = 5$ and $d \neq 2, 3, 7, 57$	[25]	1960
e	$n \neq mb$	$g = 6$ and $d = 11$	[41, 28]	1966, 1989
f	$n \neq mb$	$g = 6$ and $d \neq p^\alpha + 1$ and $d \equiv 2, 3 \mod 4$ and $d \neq a^2 + b^2 + 1$	[11, 41]	1949, 1966
g	$n \neq mb + 1$	$g = 5$	[10]	1967
h	$n \neq mb + 1$	$g = 7$	[18]	1999
i	$n \neq mb + 1$	$g = 2k + 1 \geq 5$ and $d \geq 3$	[4]	1981
j	$n \neq mb + 1$	$g = 2k \geq 6$ and $d \geq 3$	[7]	1980
k	$n \neq mb + 2$	$g = 5$ and $d = 5, 6, 7, 8, 9, 10, 11$	[18]	1999
l	$n \neq mb + 2$	$g = 6$ and $d \equiv 5, 7 \mod 8$	[7]	1980
m	$n \neq mb + 2$	$g = 2k \geq 8, d \geq 3$	[7]	1980
n	$n \neq mb + 2$	$g = 5$ and $d = 2k + 1 \geq 3$ and $d \neq a^2 + a - 1$ and $d \neq a^2 + a + 3$	[27]	1981
o	$n = mb + 2 = 24$	$d = 3, g = 7$	[30]	1960
p	$n = mb + 12 = 58$	$d = 3, g = 9$	[6, 9]	1980, 1995
q	$n = mb + 8 = 70$	$d = 3, g = 10$	[1, 34]	1972, 1980
r	$n = mb + 18 = 112$	$d = 3, g = 11$	[2, 31]	1973, 1998
s	$n = mb + 2 = 19$	$d = 4, g = 5$	[38]	1964
t	$n = mb + 14 = 67$	$d = 4, g = 7$	[22]	2011
u	$n = mb + 4 = 30$	$d = 5, g = 5$	[44]	1973
v	$n = mb + 3 = 40$	$d = 6, g = 5$	[36]	1979
w	$n = mb + 4 = 90$	$d = 7, g = 6$	[35]	1981
x	$n \geq mb + 12 = 202$	$d = 3, g = 13$	[31]	1998
y	$n \geq mb + 6 = 260$	$d = 3, g = 14$	[22]	2011
	$n \geq mb + 8 = 262$	$d = 3, g = 14$	*	2025
	$n \geq mb + 6 = 388$	$d = 3, g = 15$	*	2025
	$n \geq mb + 4 = 770$	$d = 3, g = 17$	*	2025
	$n \geq mb + 4 = 165$	$d = 4, g = 9$	*	2025
	$n \geq mb + 4 = 110$	$d = 5, g = 7$	*	2025
	$n \geq mb + 4 = 69$	$d = 8, g = 5$	*	2025

Table 1: Lower bound results for the order of cages, $n = n(d, g)$, in the literature. Here, p is a prime number, a, b and k are integers and α is a positive integer. The Moore bound is $mb = M(d, g)$. New results reported in this article have an asterisk in the reference column. First column is and identification label for reference in Table 2.

$d\backslash g$	5		6		7		8		9		10		11	
3	10	10	14	14	24	24	30	30	58	58	70	70	106✗	112
	10^{a}	10	14^{b}	14	24^{o}	22	30^{b}	30	58^{cp}	46	70^{cq}	62	112^{cr}	94
4	19	19	26	26	61✗	67	80	80	165✓	270	-	320	-	713
	19^{ds}	17	26^{b}	26	67^{cit}	53	80^{b}	80	163^{ci}	161	245^{cjm}	242	487^{ci}	485

$d\backslash g$	12		13		14		15		16		17	
3	126	126	200✗	272	262✓	384	388✓	620	514	936	770✓	2048
	126^{b}	126	202^{cx}	190	260^{cmy}	254	384^{c}	382	514^{cm}	510	768^{c}	766
4	-	728	-	-	-	-	-	-	-	-	-	-
	728^{b}	728	1459^{ci}	1457	2189^{cjm}	2186	4375^{ci}	4373	6563^{cjm}	6560	13123^{ci}	13121

$d\backslash g$	5		6		7		8	
5	30	30	42	42	110✓	152	-	170
	30^{du}	26	42^{b}	42	108^{c}	106	170^{b}	170
6	40	40	62	62	-	294	-	312
	40^{dgv}	37	62^{b}	62	189^{ci}	187	312^{b}	312
7	50	50	90	90	-	632	-	658
	50^{a}	50	90^{flw}	86	304^{c}	302	518	518
8	69✓	80	114	114	-	774	-	800
	68^{dgk}	65	114^{b}	114	459^{ci}	457	800^{b}	800
9	84✗	96	-	146	-	1104	-	1170
	86^{dk}	82	146^{b}	146	660^{c}	658	1170^{b}	1170

Table 2: Bounds for (d, g)-cages. In each cell, the lower right number is the Moore bound; lower left: best lower bound in literature (superscripts refer to results in Table 1); upper right: best upper bound in the literature according to [13, 21, 20]; upper left: our best lower bound.

The cases that took up most time are presented in Table 3. All other cases that did finish, had a combined running time of 41.89 minutes; discrepancies with the times reported in [16] are due to a recalculation of some cases with the latest fine-tuned optimization parameters for our algorithms (three cases ran slightly slower though).

(d, g, n)	time
$(3, 16, 512)$	2.83 months
$(7, 6, 90)$	2.14 months
$(3, 9, 58)$	15.7 days
$(8, 5, 68)$	14.4 days
$(3, 13, 198)$	8.97 days
$(3, 11, 104)$	4.53 days
$(4, 7, 60)$	3.77 days
$(3, 14, 260)$	1.27 days

(d, g, n)	time
$(4, 9, 164)$	18.6 h
$(3, 16, 510)$	16.0 h
$(3, 15, 386)$	12.7 h
$(3, 9, 56)$	8.87 h
$(4, 7, 59)$	5.92 h
$(3, 13, 196)$	5.65 h
$(5, 7, 108)$	5.42 h
$(3, 11, 102)$	2.99 h

(d, g, n)	time
$(9, 5, 82)$	1.60 h
$(3, 17, 768)$	33.8 min
$(4, 7, 58)$	24.7 min
$(3, 14, 258)$	21.6 min
$(8, 5, 67)$	19.2 min
$(3, 9, 54)$	13.2 min
$(7, 6, 88)$	11.3 min
$(4, 9, 163)$	10.1 min

Table 3: Running times for most time-consuming cases.

Acknowledgments. We are grateful to Geoffrey Exoo for several recommendations that improved our algorithms, to David Flores for pointing out a mistake in a previous version of the paper and to Banff International Research Station (BIRS) for its support to at-

tend the excellent meeting in 2023 (23w5125: *Extremal Graphs arising from Designs and Configurations*) that allowed us to have many fruitful exchanges with Geoffrey Exoo. We are also grateful to the anonymous referees for their valuable suggestions, which helped improve the presentation of this work and motivated the inclusion of Section 4.1 at revision time. The authors received support from CONAHCYT grant 799875, CONAHCYT grant A1-S-45528, PAPIIT grant IN113324, SECIHTI grant CBF 2023-2024-552.

References

[1] A. T. Balaban. *A trivalent graph of girth ten.* J. Combinatorial Theory Ser. B **12** (1972) 1–5.

[2] A. T. Balaban. *Trivalent graphs of girth nine and eleven, and relationships among cages.* Rev. Roumaine Math. Pures Appl. **18** (1973) 1033–1043.

[3] E. Bannai and T. Ito. *On finite Moore graphs.* J. Fac. Sci. Univ. Tokyo Sect. IA Math. **20** (1973) 191–208.

[4] E. Bannai and T. Ito. *Regular graphs with excess one.* Discrete Math. **37** (1981) 147–158. doi:10.1016/0012-365X(81)90215-6.

[5] C. T. Benson. *Minimal regular graphs of girths eight and twelve.* Canad. J. Math. **18** (1966) 1091–1094.

[6] N. L. Biggs and M. J. Hoare. *A trivalent graph with* 58 *vertices and girth* 9. Discrete Math. **30** (1980) 299–301.

[7] N. L. Biggs and T. Ito. *Graphs with even girth and small excess.* Math. Proc. Cambridge Philos. Soc. **88** (1980) 1–10. doi:10.1017/S0305004100057303.

[8] J. A. Bondy and U. S. R. Murty. *Graph theory*, volume 244 of *Graduate Texts in Mathematics*. Springer, New York, 2008.

[9] G. Brinkmann, B. D. McKay and C. Saager. *The smallest cubic graphs of girth nine.* Combin. Probab. Comput. **4** (1995) 317–329. doi:10.1017/S0963548300001693.

[10] W. G. Brown. *On the non-existence of a type of regular graphs of girth* 5. Canad. J. Math. **19** (1967) 644–648. doi:10.4153/CJM-1967-058-3.

[11] R. H. Bruck and H. J. Ryser. *The nonexistence of certain finite projective planes.* Canad. J. Math. **1** (1949) 88–93. doi:10.4153/CJM-1949-009-2.

[12] C. Cedillo, D. López, R. MacKinney-Romero, M. A. Pizaña, I. A. Robles and R. Villarroel-Flores. *YAGS - Yet Another Graph System, Version 0.0.6*, 2025. http://xamanek.izt.uam.mx/yags/+.

[13] K. Coolsaet, S. D'hondt and J. Goedgebeur. *House of graphs 2.0: A database of interesting graphs and more.* Discrete Appl. Math. **325** (2023) 97–107. Accessed: Jan 20, 2026. https://houseofgraphs.org/meta-directory/cages+.

[14] R. M. Damerell. *On Moore graphs.* Proc. Cambridge Philos. Soc. **74** (1973) 227–236. doi:10.1017/s0305004100048015.

[15] C. M. De la Cruz and M. A. Pizaña. *Choosing and Cages (code and manual)*, 2025. http://xamanek.izt.uam.mx/map/cages/+.

[16] C. M. De la Cruz and M. A. Pizaña. *New lower bounds for the order of cages*. Procedia Comput. Sci. **273** (2025) 479–484. doi:10.1016/j.procs.2025.10.335.

[17] P. Erdős and H. Sachs. *Reguläre graphen gegebener taillenweite mit minimaler knotenzahl*. Wissenschaftliche Zeitschrift der Martin-Luther-Universität Halle-Wittenberg, Mathematisch-Naturwissenschaftliche Reihe **12** (1963) 251–258.

[18] L. Eroh and A. Schwenk. Cages of girth 5 and 7. In *Proceedings of the Thirtieth Southeastern International Conference on Combinatorics, Graph Theory, and Computing (Boca Raton, FL, 1999)*, volume 138, pages 157–173, 1999.

[19] L. Euler. *Solutio problematis ad geometriam situs pertinentis*. Commentarii Academiae Scientiarum Imperialis Petropolitanae **8** (1741) 128–140.

[20] G. Exoo, J. Goedgebeur, J. Jooken, L. Stubbe and T. Van den Eede. *New small regular graphs of given girth: the cage problem and beyond*. arXiv preprint (2025). arXiv:2511.07247.

[21] G. Exoo and R. Jajcay. *Dynamic cage survey*. Electron. J. Combin. **#DS16** (2013) 55. doi:10.37236/37.

[22] G. Exoo, B. D. McKay, W. Myrvold and J. Nadon. *Computational determination of* $(3,11)$ *and* $(4,7)$ *cages*. J. Discrete Algorithms **9** (2011) 166–169. doi:10.1016/j.jda.2010.11.001.

[23] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.14.0*, 2024. http://www.gap-system.org.

[24] F. Harary. *Graph theory*. Addison-Wesley Publishing Co., Reading, Mass.-Menlo Park, Calif.-London, 1969.

[25] A. J. Hoffman and R. R. Singleton. *On Moore graphs with diameters* 2 *and* 3. IBM J. Res. Develop. **4** (1960) 497–504. doi:10.1147/rd.45.0497.

[26] A. Hulpke. *An overview of computational group theory* (2010). Accessed: August 12, 2022. https://www.math.colostate.edu/~hulpke/talks/CGTIntro.pdf.

[27] P. Kovács. *The nonexistence of certain regular graphs of girth* 5. J. Combin. Theory Ser. B **30** (1981) 282–284. doi:10.1016/0095-8956(81)90045-9.

[28] C. W. H. Lam, L. Thiel and S. Swiercz. *The nonexistence of finite projective planes of order* 10. Canad. J. Math. **41** (1989) 1117–1123. doi:10.4153/CJM-1989-049-4.

[29] S. Linton. Finding the smallest image of a set. In *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation*, ISSAC '04, page 229–234, New York, NY, USA, 2004. Association for Computing Machinery.

[30] W. F. McGee. *A minimal cubic graph of girth seven*. Canad. Math. Bull. **3** (1960) 149–152.

[31] B. McKay, W. Myrvold and J. Nadon. Fast backtracking principles applied to find new cages. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (San Francisco, CA, 1998)*, pages 188–191. ACM, New York, 1998.

[32] B. D. McKay and A. Piperno. *Practical graph isomorphism, II.* J. Symbolic Comput. **60** (2014) 94–112. doi:10.1016/j.jsc.2013.09.003.

[33] B. D. McKay and A. Piperno. *nauty and Traces* (2026). Accessed: Jan 11, 2026. https://pallini.di.uniroma1.it.

[34] M. O'Keefe and P. K. Wong. *A smallest graph of girth* 10 *and valency* 3. J. Combin. Theory Ser. B **29** (1980) 91–105. doi:10.1016/0095-8956(80)90046-5.

[35] M. O'Keefe and P. K. Wong. *The smallest graph of girth* 6 *and valency* 7. J. Graph Theory **5** (1981) 79–85. doi:10.1002/jgt.3190050105.

[36] M. O'Keefe and P. K. Wong. *A smallest graph of girth* 5 *and valency* 6. J. Combin. Theory Ser. B **26** (1979) 145–149. doi:10.1016/0095-8956(79)90052-2.

[37] C. Pech and S. Reichard. *Enumerating Set Orbits*, in Algorithmic Algebraic Combinatorics and Gröbner Bases, pages 137–150. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009 doi:10.1007/978-3-642-01960-9_4.

[38] N. Robertson. *The smallest graph of girth* 5 *and valency* 4. Bull. Amer. Math. Soc. **70** (1964) 824–825. doi:10.1090/S0002-9904-1964-11250-7.

[39] Á. Seress. *Permutation Group Algorithms*. Cambridge Tracts in Mathematics. Cambridge University Press, 2003 doi:10.1017/CBO9780511546549.

[40] C. C. Sims. Computational methods in the study of permutation groups. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 169–183. Pergamon, 1970. doi:10.1016/B978-0-08-012975-4.50020-5.

[41] R. Singleton. *On minimal graphs of maximum even girth.* J. Combinatorial Theory **1** (1966) 306–332. doi:10.1016/S0021-9800(66)80054-6.

[42] W. T. Tutte. *A family of cubical graphs.* Proc. Cambridge Philos. Soc. **43** (1947) 459–474. doi:10.1017/s0305004100023720.

[43] O. Veblen and W. H. Bussey. *Finite projective geometries.* Trans. Am. Math. Soc. **7** (1906) 241–259. http://www.jstor.org/stable/1986438.

[44] G. Wegner. *A smallest graph of girth* 5 *and valency* 5. J. Combin. Theory Ser. B **14** (1973) 203–208. doi:10.1016/0095-8956(73)90003-8.