Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2025) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

XIII Latin American Algorithms, Graphs, and Optimization Symposium (LAGOS 2025)

New lower bounds for the order of cages

Claudia Marlene de la Cruz[1,2,3], Miguel Pizaña[1,3,*]

Universidad Autónoma Metropolitana, Av. San Rafael Atlixco 186, Mexico City 09340, Mexico.

Abstract

A (d, g)-*cage* is a d-regular graph of girth g with minimum order. We denote this order by $n(d, g)$. Using group theoretic results and exhaustive computational searches, we improved 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.

Keywords: cages; algorithms; choosing; groups.

1. Introduction

A (d, g)-*graph* is a (simple, finite) d-regular graph of girth g. A (d, g)-*cage* is a (d, g)-graph of minimum order. We denote the order of a (d, g)-cage by $n(d, g)$. Cages were introduced by Tutte in [22] and have been extensively studied since then (see [9]). Cages with $d = 2$ (cycles), $g = 3$ (complete graphs) or $g = 4$ (complete bipartite graphs) are generally considered to be trivial and hence, we usually study only cages with $d \geq 3$ and $g \geq 5$.

Finding cages or even determining the value of $n(d, g)$ in general is a difficult problem: Besides 3 infinite families of known cages (for $g = 6, 8$ and 12), only eleven other cases are settled, namely: $(3, g)$ for $g = 5, 7, 9, 10, 11$; $(d, 5)$ for $d = 3, 4, 5, 6, 7$ (note that $(3, 5)$ appears in both lists); $(7, 6)$ and $(4, 7)$). For other cases, we only know bounds on $n(d, g)$. Any (d, g)-graph gives an upper bound on $n(d, g)$ and record holders for these upper bounds are carefully

* Corresponding author.
E-mail addresses: clau_mar@ciencias.unam.mx (Claudia Marlene de la Cruz)., mpizana@gmail.com (Miguel Pizaña).

URL: http://xamanek.izt.uam.mx/map (Miguel Pizaña)

[1] Equal contribution

[2] Partially supported by CONAHCYT, grant 799875.
[3] Partially supported by CONAHCYT, grant A1-S-45528.

E-mail address: mpizana@gmail.com

accounted in the literature. In the case of lower bounds there are both, theoretical results [12, 21, 5, 1, 7, 3, 14, 2, 8] and results based on exhaustive computational searches [17, 18, 19, 4, 15, 10].

A fundamental lower bound is the celebrated *Moore bound* (as reported by [12]): 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:

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

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*.

Computational techniques dealing with lower bounds involve exhaustive searches using backtracking. Backtracking is a standard programming technique, but approaches, implementation details and pruning rules are very important for performance when dealing with difficult search problems such as finding cages. Our main contribution here is the heavy use symmetries and group theoretic algorithms for conflict reduction and case reduction, but other techniques also play a major part in performance including: Speculative exploration for conflict selection, dynamic programming, lazy evaluation and reimplementation of some standard GAP procedures.

Our code was implemented in GAP [11] and YAGS [6]. We also used *nauty* [16] for computing the automorphism groups of graphs, and the included programs *geng* and *pickg* for generation of possible *excess graphs*. We heavily use the procedures `Stabilizer(G,x)` and `RepresentativeAction(G,x,y)` (see GAP's manual), however, GAP's implementations do more than needed. For instance, `Stabilizer(G,x)` computes a complete stabilizer chain instead of just one stabilizer. In the case of the other GAP procedure `RepresentativeAction(G,x,y)`, which we call `Transporter(G,x,y)`, we reimplemented it, to use path compression in the orbit-stabilizer trees and lazy evaluation for the products of permutations so that the products get done only when needed. Therefore, for performance reasons, we needed to reimplement those procedures using the descriptions of the algorithms in [20] and [13]. We used dynamic programming by storing all relevant results (orbit-stabilizer trees, stabilizers of groups, etc) to avoid recalculations.

2. Searching for cages

We searched for cages using backtracking. We use a stack of pending cases (feasible solutions, graphs) to keep track of the computation. A *feasible solution* is a graph H with $\Delta(H) \leq d$, $\mathrm{Girth}(H) \geq g$ and $|H| = n$. A *conflict* is a vertex $x \in H$ such that $\deg(x) < d$ and a *solution* for a conflict is a lists of graphs (new feasible solutions) $H_1, H_2, \ldots, H_s$ obtained from H by completing the neighborhood of x in H in all possible ways up to symmetries of H. The initial contents of the stack is simply the disjoint union of a Moore tree, `T`, and a discrete graph, `Ex`, on $n - mb$ vertices. The contents of the main cycle of the algorithm may be expressed as follows.

1. Remove a graph H from the stack.
2. Compute the set of conflicts, `conf` up to symmetries of H.
3. Select a conflict, `x ∈ conf`.
4. Solve the conflict, `x`.
5. Put the new feasible solutions on the stack.

Our innovations here, are in the conflict selection and conflict solution procedures.

To solve a conflict we compute $\binom{S}{k}_G$: Given a set S, an integer k and a permutation group G acting on S, we denote by $\binom{S}{k}_G$ the set of all k-subsets of S up to symmetries in G (more details in the next section). Hence, if we want to

compute all possible completions of the neighbors of x, we take:

$\mathtt{k} = d - \deg(\mathtt{x})$,

$\mathtt{G} = \mathtt{Stabilizer(AutomorphismGroup(H),x)}$,

$\mathtt{S} = \{y \in V(H) : x \neq y, \deg(y) < d, \mathrm{dist}(x, y) > g - 1\}$.

In this way, for each $Y_i \in \binom{S}{k}_G$, we obtain a new graph $H_i = H \cup \{xy : y \in Y_i\}$. We mention here that our procedure for computing $\binom{S}{k}_G$ also uses backtracking and that it allows for a user-provided hook procedure, `check(Y)`, for checking additional properties that the user may require. `check(Y)` is called whenever a feasible solution $Y \subseteq S$ is extended, and the feasible solution is pruned when `check(Y)` returns false. In our case, the `check(Y)` is used to verify that no short cycles are formed by the new candidate edges $\{xy : y \in Y\}$.

When selecting a conflict, we always select the conflict x with minimum number of graphs $H_1, H_2, \ldots, H_s$ in its solution. Instead of computing all possible completions for the neighborhoods of x for every conflict x at a given time, we compute one completion for each conflict at a time and hence the first conflict that fails to have an additional completion is certainly one with a minimum number of graphs in its solution.

3. Groups and Choosing

Given a set of integers $X = \{x_1, x_2, \ldots, x_s\}$, we say that it is in *sorted presentation* if $x_k < x_{k+1}$ for all k. The *maximum element* of X is denoted by $\max X$. We define $X_{<z} = \{x \in X : x < z\}$. Given two sets in sorted presentation $X = \{x_1, x_2 \ldots, x_s\}$ and $Y = \{y_1, y_2 \ldots, y_t\}$, we say that Y is a *prefix* of X, denoted as $Y \preccurlyeq X$, if $Y \subseteq X$ and $\max Y < x$ for all $x \in X \setminus Y$.

Let N be a positive integer, $\Omega = \{1, 2, \ldots, N\}$, $S \subseteq \Omega$ and $k \leq |S|$. We denote by $\binom{S}{k}$ the set of all k-subsets of S, that is $\binom{S}{k} = \{X \subseteq S : |X| = k\}$. Now, let $\mathcal{S}_\Omega$ be the symmetric group on the elements of Ω, and let G be a subgroup, $G \leq \mathcal{S}_\Omega$. Permutations $g \in \mathcal{S}_\Omega$ are bijective functions $g : \Omega \to \Omega$, but we use exponential notation for the (right) action of G on Ω, hence for $x \in \Omega$ and $g \in G$, we define $x^g = g(x)$.

Now assume S is G-invariant, that is, $S^G = \{s^g : g \in G\} = S$. Then G also acts on S and this induces an action of G on $\binom{S}{k}$, namely, for $g \in G$ and $X \in \binom{S}{k}$, we define $X^g = \{x^g : x \in X\}$. This in turns, defines an equivalence relation on $\binom{S}{k}$, namely, $X \sim Y$ if and only if $\exists\, g \in G$ such that $Y = X^g$. The corresponding partition is then $\binom{S}{k}/\sim$. We denote by $\binom{S}{k}_G$ the set resulting from selecting, from each equivalence class of, $\binom{S}{k}/\sim$, the minimum element under the lexicographic order.

Example: Let $N = 7$, $S = \{1, 4, 5, 7\}$, $G = \langle\, (1\,4)(5\,7), (1\,4\,5\,7) \,\rangle$ (the dihedral group).
Then $\binom{S}{2}_G = \{\{1, 4\}, \{1, 5\}\}$ and $\binom{S}{3}_G = \{\{1, 4, 5\}\}$.

The naive approach for computing $\binom{S}{k}_G$ is to compute $\binom{S}{k}$ first and then select a representative for each equivalence class in $\binom{S}{k}/\sim$. This approach is unfeasible in general since $\binom{S}{k}$ may be huge even in cases when $\binom{S}{k}_G$ is very small.

Instead, the general idea for computing $\binom{S}{k}_G$ 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. For this, we need to define a precise strict partial order among subsets of Ω as follows:

Definition 3.1. *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 $Y \preccurlyeq X$ implies $Y \not< X$ and $X \not< Y$. Hence this strict partial order is similar, but not equal to the standard lexicographic order, which is a strict total order. With this order defined, the pruning condition we need is simply:

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

Again, a direct (naive) implementation of the previous code is also not feasible in general, since the group G may be huge and hence, iterating g over the elements of G is not feasible. Instead, we implement that function much more efficiently by using the following theorems.

Theorem 3.2. *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* $X_{<z} \subseteq Y$.

Theorem 3.3. *Let $X \subseteq S$, $g \in G$ and $x_n = \max X$. Then, the following conditions are equivalent.*

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

Theorem 3.4. *$Y \preccurlyeq X$ and $Y^g < Y \Rightarrow X^g < X$.*

Theorem 3.5. *Let $X \subseteq S$, $x_n = \max X$ and $g \in G$. Assume $X^g < X$. Let z be as in Theorem 3.3(4). Furthermore, assume that for every proper prefix $Y \prec X$ we have that $Y^g \not< Y$. Then one of the following conditions hold:*

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

Many other important details will presented in the corresponding full article.

4. Experimental results

Our algorithm ran on a single processor, Intel Core i7 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 2 months, but many processes were aborted after 2 days when no significant progress was achieved in that time.

Table 1 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 (and our patience ran out before the algorithm did) 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 (run in a few weeks on a mixture of machines) to obtain the lower bound. We currently do not have access to such computing power.

In all cases, we started with the Moore bound (even if better bounds were known) and try higher and higher bounds when feasible. For instance, our bound $n(4, 9) \geq 165$ means that all values from 161 to 164 where 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 where able to finish the search for the case and find all the known cages for that parameters (all the eighteen $(3, 9)$-cages on 58 vertices in this case). The cases that took up most time are presented in Table 2. All other cases that finished, had a combined running time of 1.03 hours.

Acknowledgment: 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 economical support to attend the excellent meeting in 2023 (23w5125: *Extremal Graphs arising from Designs and Configurations*) that allowed us to have many fruitful exchanges with Geoffrey Exoo.

$d\backslash g$	5		6		7		8		9		10		11	
3	10	10	14	14	24	24	30	30	58	58	70	70	106✗	112
	10	10	14	14	24	22	30	30	58	46	70	62	112	94
4	19	19	26	26	61✗	67	80	80	165✓	275	-	384	-	-
	19	17	26	26	67	53	80	80	163	161	245	242	487	485

$d\backslash g$	12		13		14		15		16		17	
3	126	126	200✗	272	262✓	384	388✓	620	512✗	960	770✓	2176
	126	126	202	190	260	254	384	382	514	510	768	766
4	-	728	-	-	-	-	-	-	-	-	-	-
	728	728	1459	1457	2189	2186	4375	4373	6563	6560	13123	13121

$d\backslash g$	5		6		7		8	
5	30	30	42	42	110✓	152	-	170
	30	26	42	42	108	106	170	170
6	40	40	62	62	-	294	-	312
	40	37	62	62	189	187	312	312
7	50	50	90	90	-	-	-	672
	50	50	90	86	304	302	518	518
8	69✓	80	114	114	-	-	-	800
	68	65	114	114	459	457	800	800
9	84✗	96	-	146	-	1152	-	1170
	86	82	146	146	660	658	1170	1170

Table 1. Bounds for (d, g)-cages. In each cell, the lower right number is the Moore bound; lower left: best lower bound in literature; upper right: best upper bound in the literature according to [9]; upper left: our best lower bound.

(d, g, n)	time
$(7, 6, 90)$	1.99 months
$(4, 7, 60)$	1.01 months
$(3, 9, 58)$	15.8 days
$(8, 5, 68)$	14.9 days
$(3, 13, 198)$	8.30 days
$(3, 14, 260)$	4.60 days
$(3, 11, 104)$	4.48 days

(d, g, n)	time
$(4, 9, 164)$	4.12 days
$(3, 9, 56)$	1.95 days
$(5, 7, 108)$	1.37 days
$(3, 16, 510)$	16.0 h
$(3, 15, 386)$	12.7 h
$(4, 7, 59)$	5.92 h
$(3, 13, 196)$	5.65 h

(d, g, n)	time
$(3, 11, 102)$	2.99 h
$(9, 5, 82)$	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

Table 2. Running times for most time-consuming cases.

References

[1] Bannai, E., Ito, T., 1973. On finite Moore graphs. J. Fac. Sci. Univ. Tokyo Sect. IA Math. 20, 191–208.

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

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

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

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

[6] Cedillo, C., López, D., MacKinney-Romero, R., Pizaña, M.A., Robles, I.A., Villarroel-Flores, R., 2019. YAGS - Yet Another Graph System, Version 0.0.6. URL: http://xamanek.izt.uam.mx/yags/.

[7] Damerell, R.M., 1973. On Moore graphs. Proc. Cambridge Philos. Soc. 74, 227–236. doi:10.1017/s0305004100048015.

[8] Eroh, L., Schwenk, A., 1999. Cages of girth 5 and 7, in: Proceedings of the Thirtieth Southeastern International Conference on Combinatorics, Graph Theory, and Computing (Boca Raton, FL, 1999), pp. 157–173.

[9] Exoo, G., Jajcay, R., 2013. Dynamic cage survey. Electron. J. Combin. DS16, 55. doi:10.37236/37.

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

[11] GAP, 2019. GAP – Groups, Algorithms, and Programming, Version 4.10. The GAP Group. URL: http://www.gap-system.org.

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

[13] Hulpke, A., 2010. An overview of computational group theory. URL: https://www.math.colostate.edu/~hulpke/talks/CGTIntro.pdf. retrieved on August 10, 2022.

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

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

[16] McKay, B.D., 1990. *nauty* User's Guide (Version 2.4). Technical Report TR-CS-90-02. Australian National University, Computer Science Department. URL: http://cs.anu.edu.au/~bdm/nauty/.

[17] O'Keefe, M., Wong, P.K., 1979. A smallest graph of girth 5 and valency 6. J. Combin. Theory Ser. B 26, 145–149. doi:10.1016/0095-8956(79)90052-2.
[18] O'Keefe, M., Wong, P.K., 1980. A smallest graph of girth 10 and valency 3. J. Combin. Theory Ser. B 29, 91–105. doi:10.1016/0095-8956(80)90046-5.
[19] O'Keefe, M., Wong, P.K., 1981. The smallest graph of girth 6 and valency 7. J. Graph Theory 5, 79–85. doi:10.1002/jgt.3190050105.
[20] Seress, Á., 2003. Permutation Group Algorithms. Cambridge Tracts in Mathematics, Cambridge University Press. doi:10.1017/CBO9780511546549.
[21] Singleton, R., 1966. On minimal graphs of maximum even girth. Journal of Combinatorial Theory 1, 306–332. doi:10.1016/S0021-9800(66)80054-6.
[22] Tutte, W.T., 1947. A family of cubical graphs. Proc. Cambridge Philos. Soc. 43, 459–474. doi:10.1017/s0305004100023720.