Journal of Discrete Algorithms 9 (2011) 166–169

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Computational determination of (3, 11) and (4, 7) cages

Geoffrey Exoo[a], Brendan D. McKay[b,*,1], Wendy Myrvold[c], Jacqueline Nadon[c]

[a] Department of Mathematics and Computer Science, Indiana State University, Terre Haute, IN 47809, United States
[b] School of Computer Science, Australian National University, Canberra, ACT 0200, Australia
[c] Department of Computer Science, University of Victoria, Victoria, B.C., Canada V8W 3P6

ARTICLE INFO

Article history:
Received 24 March 2009
Accepted 22 July 2009
Available online 4 November 2010

Keywords:
Cage
Regular graph
Girth

ABSTRACT

A (k, g)-*graph* is a k-regular graph of girth g, and a (k, g)-*cage* is a (k, g)-graph of minimum order. We show that a $(3, 11)$-graph of order 112 found by Balaban in 1973 is minimal and unique. We also show that the order of a $(4, 7)$-cage is 67 and find one example. Finally, we improve the lower bounds on the orders of $(3, 13)$-cages and $(3, 14)$-cages to 202 and 260, respectively. The methods used were a combination of heuristic hill-climbing and an innovative backtrack search.

© 2010 Elsevier B.V. All rights reserved.

1. Introduction

A k-*regular* graph is one in which every vertex has degree k. The *girth* of a graph is the length of a shortest cycle. A (k, g)-*graph* is a regular graph of degree k and girth g, and a (k, g)-*cage* is a (k, g)-graph of minimum possible order.

The problem of determining this order and identifying the corresponding cages has been extensively studied. See [3] for a current survey. The cases where the order of the cage was known precisely before this work can be summarized as follows.

1. Regular graphs of degree 3 for girths up to 12;
2. Girth 5 graphs for degrees up to 7;
3. Girth 6, 8 and 12 graphs for degree one more than a prime power;
4. The case of degree 7 and girth 6.

In this note, we add the case of degree 4 and girth 7 to the list, and also show that the $(3, 11)$-cage is unique.

The order, but not the uniqueness, of the $(3, 11)$-cage was previously announced in [5].

2. Backtrack search

The uniqueness of the $(3, 11)$-cage, and the lower bound of 67 for the $(4, 7)$-cage, was proved using the program mentioned in [5] but not described in detail there. We will provide that description here using the $(4, 7)$-cage as an example.

Consider the construction of 4-regular graphs of girth at least 7 and order $n \geqslant 53$. The vertices at distance at most 3 from some fixed vertex form a tree T with 53 vertices as in Fig. 1. (In the case of even girth we would root the tree at an edge rather than a vertex.)

* Corresponding author.
E-mail addresses: gexoo@indstate.edu (G. Exoo), bdm@cs.anu.edu.au (B.D. McKay), wendym@csc.uvic.ca (W. Myrvold).
[1] Supported by the Australian Research Council.

1570-8667/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.jda.2010.11.001

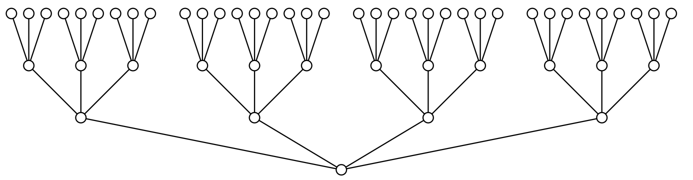
Fig. 1. The initial tree T.

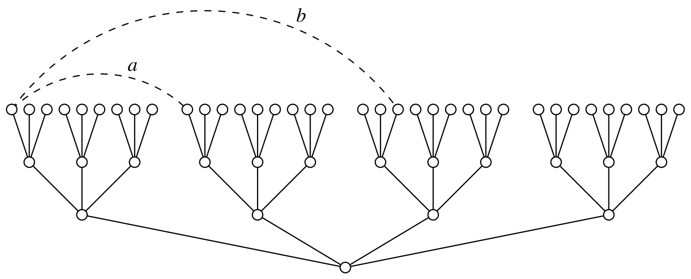

Fig. 2. Two equivalent choices.

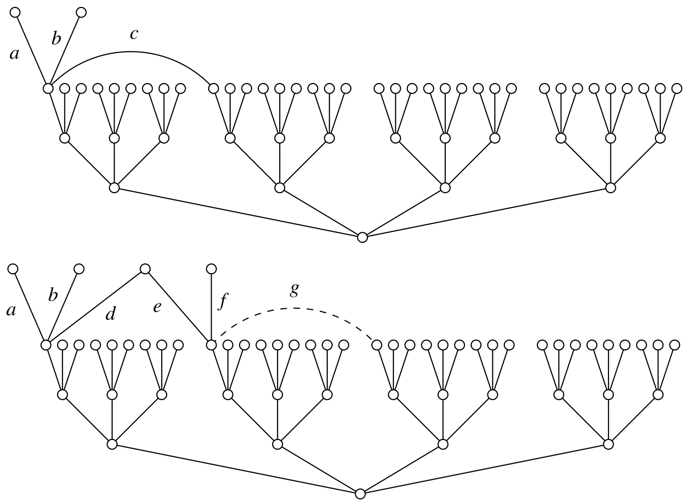

Fig. 3. Complex equivalence.

Let W be the set of $n-17$ vertices consisting of the 36 leaves of T and the $n-53$ vertices not in T. The task is thus to add additional edges within W so that the resulting graph is quartic and girth at least 7. We can do this in standard depth-first manner, starting with the tree and adding one edge at a time. For addition of a new edge vw to be valid, v and w must have degree less than 4 and be at distance at least 6; these properties are easily monitored.

At each stage in the search, we choose one vertex of degree less than 4 and try all the possibilities for joining it to other vertices. We choose the vertex whose options are the most restricted, as experiments showed this heuristic to be a good one.

Without additional improvements, this search is far too expensive due to multiple equivalent subcases. The most obvious source of equivalence is the automorphism group of the tree. Thus, the two choices a, b in Fig. 2 are clearly equivalent and there is no need to try b as the first choice once a has been tried. The simple structure of the tree allows examples of this type of equivalence to be monitored efficiently without explicit computation of automorphism groups.

However, when a moderate number of edges have been added, discovery of automorphisms is considerably more difficult. More importantly, there are equivalent subcases in the search that do not derive directly from automorphisms of any of the graphs that are constructed. Consider Fig. 3.

Suppose we have completed the part of the search that begins with the three edges a, b, c in the upper diagram of the figure. This means that, up to isomorphism, we have already found all the quartic graphs that contain the upper diagram as a subgraph. Now suppose that at a later point in the search we have added the edges a, b, d, e, f shown in the lower diagram of Fig. 3, and are about to consider further edges. If we add edge g, then we will not find any new quartic graphs, since $T + \{e, f, g\}$ is isomorphic to $T + \{a, b, c\}$. Therefore, we can mark g as ineligible. This applies whether we consider g as the very next edge to add or we consider adding it to some non-trivial extension of a, b, d, e, f. For efficiency reasons we separate these two cases.

We now define the pruning process formally. Each node of the search tree corresponds to a graph $T + E$, where E is a set of edges within W. The *subsearch rooted at* $T + E$ is the subtree of the search tree rooted at $T + E$, and $T + E$ is *completed* when we have finished the scan of that subtree.

Pruning rule. Suppose that $T + E$ is a search node and vw is an available edge (that is, v and w have degree less than 4 and are at distance at least 6 from each other). Suppose further that there is a subset $E' \subseteq E + vw$ such that $T + E'$ is isomorphic to a node which is already completed. Then the subsearch rooted at $T + E$ can avoid adding the edge vw.

The pruning rule is applied using the `nauty` graph isomorphism software [4]. Since subgraph isomorphism testing is required, the rule is very expensive compared to the time otherwise required to process one node of the search. In practice we limit it to only certain E and certain v, w. Specifically, we define two levels $\ell_1 \geqslant \ell_2 \geqslant 0$. For search nodes $T + E$ such that $|E| \leqslant \ell_2$, we apply the rule for all v, w. For $\ell_2 < |E| \leqslant \ell_1$, we apply it only to those v, w such that vw is a candidate for the very next edge to add.

For larger ℓ_1, ℓ_2, the number of nodes in the search tree is reduced but the time expended in pruning the tree is increased. So there is some optimal compromise.

A further technique is needed in very difficult cases (such as $n = 66$), for which it is desirable to divide the computation across multiple processes (perhaps on different computers). The division is accomplished in the usual fashion: for some level ℓ_0, treat the subsearchs rooted at $T + E$ for $|E| = \ell_0$ as independent. Subsearch i is assigned to process j if $i \equiv j \pmod N$, where N is the number of processes.

Each process involved in the search computes the whole search tree to level ℓ_0 and only its own subsearches at higher levels. Since the application of the pruning rule is hard to achieve across multiple processes, we use $\ell_0 > \ell_1$. Higher ℓ_0 also tends to share the load more evenly between processes. These considerations, however, imply that a large part of the tree, including all the rule applications, are repeated in every process, which would seem to prevent use of large ℓ_1, ℓ_2 due to the expensive nature of the pruning rule. The solution is to conduct the computation in two phases. In the first phase, executed on a single processor, the search tree is computed up to level ℓ_1 while applying the pruning rule. During this phase, the results of all the rule applications are recorded in an audit file. Then, in the second phase where the full search tree is computed, the pruning rule is applied at almost no cost by following the audit file. If ℓ_0 is not too high, the cost of the division of the second phase into parts is negligible.

For $n = 66$, we used $\ell_0 = 22$, $\ell_1 = 16$ and $\ell_2 = 6$. The total number of nodes in the search tree was 318,904,129,273,923. Using a large mix of computers ranging from 650 MHz to 3 GHz, the total time was 96 years (105,000 nodes per second). This cost was about 5 times the cost for $n = 65$.

Improvements to the program after the computation finished resulted in a 4-fold speedup. Nevertheless, the computation of all the cages of order 67 will not be feasible in the near future.

Exhaustive computation of the $(3, 11)$-cages, showing that the graph found by Balaban [1] is the only one, required 17 years on computers averaging about 300 MHz.

With lesser expenditure of cpu time, we also showed that a $(3, 13)$-cage has at least 202 vertices (improved from 196), and that a $(3, 14)$-cage has at least 260 vertices (improved from 256). The first result, and the bound 258 for the second, were previously announced in [5].

To emphasise the efficiency of this method, we compared it against the previously best code for cubic graphs [2]. For girth 9 and 58 vertices, our approach was 52 times faster, while for girth 11 and 104 vertices our approach was 856 times faster. All of the results we highlight in this paper were previously infeasible.

3. A graph of degree 4 and girth 7

As noted in the previous section, we showed exhaustively that there are no $(4, 7)$-graphs with 66 or fewer vertices. To complete the proof that $(4, 7)$-cages have 67 vertices, we present an example (see Fig. 4).

Our example on 67 vertices is given below as an adjacency list. The graph has an automorphism group of order four, isomorphic to $Z_2 \times Z_2$. In the action of this group on the vertices, there are 11 orbits of length 4, 11 orbits of length 2, and one fixed point.

The graph was constructed using a hill-climbing algorithm that begins with an empty graph on 67 vertices and adds edges, one at a time, while not violating the degree and girth conditions. An outline of the algorithm follows.

```
while there are vertices with degree < 4:
      while there are edges that can be added:
            for each edge that can be added
                compute the degree sum of its vertices
            pick the edge with the largest sum
            choose randomly in case of ties
```

0:	1 2 3 4	23:	7 61 43 29	46:	14 31 41 24
1:	0 5 6 7	24:	7 26 36 46	47:	15 42 58 26
2:	0 8 9 10	25:	7 63 40 34	48:	15 34 35 19
3:	0 11 12 13	26:	8 53 24 47	49:	15 39 65 29
4:	0 14 15 16	27:	8 43 45 22	50:	16 33 66 43
5:	1 17 18 19	28:	8 52 18 56	51:	16 36 54 30
6:	1 20 21 22	29:	9 23 49 59	52:	16 28 21 40
7:	1 23 24 25	30:	9 19 62 51	53:	59 66 38 26
8:	2 26 27 28	31:	9 20 46 57	54:	58 64 51 22
9:	2 29 30 31	32:	10 44 65 21	55:	39 62 33 56
10:	2 32 33 34	33:	10 50 55 17	56:	28 55 35 61
11:	3 35 36 37	34:	10 48 64 25	57:	58 31 17 40
12:	3 38 39 40	35:	11 56 48 20	58:	54 57 47 61
13:	3 41 42 43	36:	11 51 65 24	59:	64 37 29 53
14:	4 44 45 46	37:	11 59 17 45	60:	18 66 63 65
15:	4 47 48 49	38:	12 53 19 44	61:	58 44 23 56
16:	4 50 51 52	39:	12 55 49 22	62:	55 30 63 42
17:	5 57 37 33	40:	12 52 25 57	63:	25 62 60 45
18:	5 60 28 41	41:	13 46 18 64	64:	59 54 34 41
19:	5 38 30 48	42:	13 62 47 21	65:	36 32 49 60
20:	6 35 31 66	43:	13 23 27 50	66:	50 60 53 20
21:	6 32 52 42	44:	14 32 38 61		
22:	6 54 39 27	45:	14 27 37 63		

Fig. 4. Adjacency list for a $(4,7)$-graph of order 67.

```
          add the winning edge
      if all vertices have degree 4:
          save graph
          exit
      delete 1, 2 or 3 random edges
```

A few of the steps require some explanation.

1. Edges are added between vertices with maximum possible degree sum. Doing this in the early stages of the algorithm appears to be essential to finding a cage. We ran our program many times with this condition removed, without success.
2. When edges are added, a record is made of when they were added. Time is measured in trips through the outer loop.
3. When edges are chosen for deletion, it is done in one of two ways. Either the probability that an edge is chosen is proportional to its age, or it is inversely proportional to its age. These two modes are alternated, each being used for a few thousand time periods. Very recently deleted edges are not chosen for reinsertion.

A program implementing this method has been run several hundred times. It succeeds in finding a graph approximately twenty percent of the time. Each time it has succeeded, it has found the same graph.

References

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

[2] G. Brinkmann, B.D. McKay, C. Saager, The smallest cubic graphs of girth nine, Combin. Probab. Comput. 5 (1995) 1–13.

[3] G. Exoo, R. Jajcay, Dynamic cage survey, Electron. J. Combin. (2008), 48 pp., available at http://www.combinatorics.org/Surveys/ds16.pdf.

[4] B.D. McKay, Practical graph isomorphism, in: 10th Manitoba Conference on Numerical Mathematics and Computing, Winnipeg, 1980, in: Congr. Numer., vol. 30, 1981, pp. 45–87, software at http://cs.anu.edu.au/~bdm/nauty/.

[5] B.D. McKay, W. Myrvold, J. Nadon, Fast backtracking principles applied to find new cages, in: 9th Annual ACM–SIAM Symposium on Discrete Algorithms, January 1998, pp. 188–191.