Fast Backtracking Principles Applied to Find New Cages

Brendan McKay* Wendy Myrvold† Jacqueline Nadon‡

Abstract

We describe how standard backtracking rules of thumb were successfully applied to the problem of characterizing $(3,g)$-cages, the minimum order 3-regular graphs of girth g. It took just 5 days of cpu time (compared to 259 days for previous authors) to verify the $(3,9)$-cages, and we were able to confirm that $(3,11)$-cages have order 112 for the first time ever. The lower bound for a $(3,13)$-cage is improved from 196 to 202 using the same approach. Also, we determined that a $(3,14)$-cage has order at least 258.

1 Cages

In this paper, we consider finite undirected graphs. Any undefined notation follows Bondy and Murty [7]. The *girth* of a graph is the size of a smallest cycle. A (r,g)-cage is an r-regular graph of minimum order with girth g. It is known that (r,g)-cages always exist [11]. Some nice pictures of small cages are given in [9, pp. 54-58].

The classification of the cages has attracted much interest amongst the graph theory community, and many of these have special names. Our interest in cages arose from the problem of determining the structure of the most reliable networks under the all-terminal model (see [10] for a survey of network reliability, and [6] for an introduction to the synthesis question). We suspect that large girth is a critical factor in maximizing the reliability when edges are very reliable (the situation most often occurring in practical applications) and give evidence to support this for the 3-regular case [15, 16].

We concentrate on the 3-regular cages; a complete history of the results for this and other cases can be found in [20]. We summarize the status of the problem for girths three through twelve in Figure 1. The lower bound (*LB*) on the order is more fully explained in Section 3. The order of the cage equals the lower bound plus the number of extra vertices (*Extra*). Often cages were found (*Found*) but only later was it confirmed that they had minimum order (*Opt.*). The number of cages up to isomorphism is indicated in the last column (*Cages*).

For girth 13, Brinkmann, McKay, and Saager [8] indicated a lower bound of 196 vertices. We used our program to verify that at least 202 vertices are required. We also determined that at least 258 vertices are required for a $(3,14)$-cage (four more than the lower bound).

Our program took about five days to verify the eighteen cages of girth nine whereas the program **maus** [8] required 259 days on the same scale of time/mixture of machine types (both experiments were run by McKay ensuring consistency). The **maus** program took 2.5 years of cpu time to determine that there are no girth 11 cages with 104 vertices. Our approach confirmed this in 25.6 hours. This extra speed made it feasible to complete the search for girth 11 in 6.7 years of cpu time (run in a few weeks on a mixture of machines), so one of our new results is that $(3,11)$-cages have order 112. Confirming that a $(3,13)$-cage of order 200 does not exist required 2.3 cpu years. For girth 14, we ruled out 256 vertices in 18.8 hours of cpu time. Faster techniques are likely required to finish the work for girths thirteen and fourteen.

2 Backtracking Rules of Thumb

In this section we present some basic principles for designing fast exponential backtracking algorithms. The algorithms we consider are exhaustive backtracks designed to provide a particular configuration (such as a graph, a combinatorial design, or a structure in a graph such as a Hamiltonian cycle) when it exists. The exhaustive nature of the search guarantees that if termination occurs without finding the desired configuration, then none exists.

1. **Start with what you know.** Incorporating the "obvious" assumptions that a person makes when starting work on a problem can vastly decrease the backtracking time.

2. **Do strong redundancy checks near the root of the search tree, but only fast checks in other places.** Elimination of branches that are provably redundant (not leading to solutions inequivalent to those found elsewhere) can be extremely successful in reducing the size of the search tree but can greatly increase the work done for each node. A good compromise is to perform strong re-

*Australian National University.
†University of Victoria. Supported by NSERC.
‡Sandwell Construction

Girth	LB	Extra	Found	Opt.	Cages
3	4	0		[18]	K_4 [18]
4	6	0		[18]	$K_{3,3}$ [18]
5	10	0		[18]	Petersen graph [18]
6	14	0		[18]	Heawood graph [18]
7	22	2		[12]	McGee graph [19]
8	30	0		[18]	Tutte-Coexter graph [18]
9	46	12	[22, 5]	[8]	18 [8]
10	62	8	[1]	[17]	3 [21]
11	94	18	[2]	NEW	
12	126	0		[3]	[3]

Figure 1: The 3-regular cages of small girth

dundancy tests near the root of the tree (where there are usually few nodes) but only simple fast tests at other places.

3. **If there are choices to be made, select a decision with a minimum number of options.** The savings in backtracking time can often be worth the effort of computing if necessary the decision with the smallest number of options, particularly near the root of the computation tree.

4. **Abort early if possible**. This goes hand in hand with the previous recommendation. Often although a configuration is not maximal, it is possible to detect that it never can be completed. If we detect this without an exhaustive search of all the maximal completions, we can avoid a lot of work.

5. **Do as little as possible at each recursive call.** Since there are generally an exponential number of recursive calls, excessive overhead at each can make the running time impractical.

6. **Keep it simple if you can.** Be prepared to sacrifice a small decrease in efficiency for considerable gains in simplicity. Complexity in the algorithm can make it much harder to establish correctness of the code, especially for unsuccessful searches.

7. **Distribute work by sending branches of the computation tree to various machines.** If the algorithm requires more time than you have available on one machine, it may still be feasible if you can utilize several machines. Due to the independence of disjoint subtrees, these backtracking algorithms are very easy to parallelize. However, the sizes of subtrees and the availability of machines in a non-dedicated cluster are impossible to predict in advance. Our advice is to break the tree into significantly more pieces than the number of processors, then to farm the pieces out to processors as they become available. Often it is enough to cut the tree across some convenient level.

3 Application: Finding New Cages

We now describe how the principles in the previous section were very effectively applied to find new 3-regular cages. Recall that 3-regular graphs have an even number of vertices due to the elementary observation that a graph must have an even number of vertices of odd degree.

1. **What we know.** An obvious lower bound (given for example in [4]) on the order of a $(3, g)$-cage is

$$\lambda(g) = \begin{cases} 2^{(g+2)/2} - 2, & \text{if } g \text{ is even; and} \\ 3 \times 2^{(g-1)/2} - 2, & \text{if } g \text{ is odd.} \end{cases}$$

   This is because a breadth first search tree truncated to height $\lfloor (g-1)/2 \rfloor$, rooted at a vertex in the odd case and an edge in the even case, yields a tree whose vertices excepting the leaves all have degree three. Adding edges elsewhere instead would create a cycle smaller than g. If there is a cage meeting the lower bound, it is possible to add edges between the leaves to create it. To search for cages of order $2k$ greater than the lower bound, we start with this tree plus $2k$ isolated vertices.

2. **Redundancy screening.**

   The following ideas were used to eliminate obvious equivalences and were applied at all levels:

   (a) **Isolated points**. Let $x_0, \ldots x_k$ denote the set of points in the graph constructed so far which have degree zero. Adding edge (v, x_0) is obviously isomorphic to adding (v, x_i), $i = 1, ..., k$.

(b) **Leaves**. We number of the leaves of the BFS tree that we start with from left to right starting with 1 (with the tree pictured in the standard way). A vertex is *touched* if it is incident to an edge which is not one of the edges of the original BFS tree. We only add an edge incident to an untouched vertex if it has the smallest label amongst the isomorphic untouched alternatives. The isomorphic alternatives to vertex x are numbered sequentially starting with x and terminating at a precomputed vertex number $end\_iso[x]$. For example, if the desired girth is seven, the values of end_iso are as pictured in Figure 2.

Nauty [14], a general program for graph isomorphism, was used to aid in the detection and elimination near the root of the search tree of branches which would not result in new cages. The ideas we applied are more generally applicable so we describe our technique in more detail.

Each branch at level k of the search tree is associated with a graph consisting of the initial tree as described in point 1 plus k *extra* edges. The children are ordered and each is associated with a graph having one additional edge. The path from the root of the computation tree to a particular node can be indicated by listing in sequence the child to select at each level, and we label the node with this sequence.

We maintain the property that the search rooted at a particular node enumerates all cages which contain the associated graph as a subgraph which have not been enumerated by a (lexicographically) earlier portion of the search tree. This is the same as insisting that we only ignore branches if the resulting cages would have been enumerated earlier assuming the code is executed on a single machine. When only a small number of extra edges have been added, it is sufficient to check whether any of the graphs arising from adding each subset of the extra edges is isomorphic to the graph associated with an earlier branch of the search tree. The situation becomes more complicated as more edges are added as there can be more than one way to select the initial tree structure described in point 1.

3. **A minimum choice decision**.

   The decision to be made is which edge should be added to the current graph. For each vertex of degree less than three, there are a fixed number of legal options for adding another edge without creating a cycle that is too small. We compute the number of choices for each, and add one additional edge incident to the one with a minimum number of options. This simple idea is one of the factors contributing to the significant speedup over the previous approach [8].

4. **Early abortion**. Although it may still be possible to add edges to the graph we have created so far, if we have a vertex of degree $3 - k$ with fewer than k legal choices for an incident edge, it is clear that the graph cannot be completed to a cage.

5. **Do as little as possible at each recursive call.** We maintain a distance matrix indexed by the vertices of degree less than three and this is used to determine the edges which may be added without violating the girth. We need this information to make our minimum choice decision. This data structure has the attractive property of decreasing in size at the deeper levels of the computation tree.

   A trick we use to avoid considering equivalent configurations is to mark invalid edges whose distance is at least $g - 1$ with the value $g - 2$. With an appropriate distance "algebra", we can prove this has the desired effect.

6. **Keeping it simple.** Our data structures in particular are very simple (as described above). Further, the data structure maintenance is completed in a straightforward manner.

7. **Work distribution.** For girth 9, the program was fast enough to be run on a single computer. For larger girth, subproblems were automatically distributed using *autoson* [13]. Hence we were able to obtain several years of computer time in just a couple of weeks.

4 Conclusions and Future Research

Computer validation of mathematical results is still a relatively new phenomenon. Unlike a traditional proof, it is not possible to check the results with pencil and paper. Many stages of the research are subject to error: errors in the theory, the program, the machine computation, and the compiler can all invalidate the results.

If we instead view the process as an experimental science, we can focus on providing the information required for the experiment to be successfully reproduced rather than the mathematical conclusions. In this case,

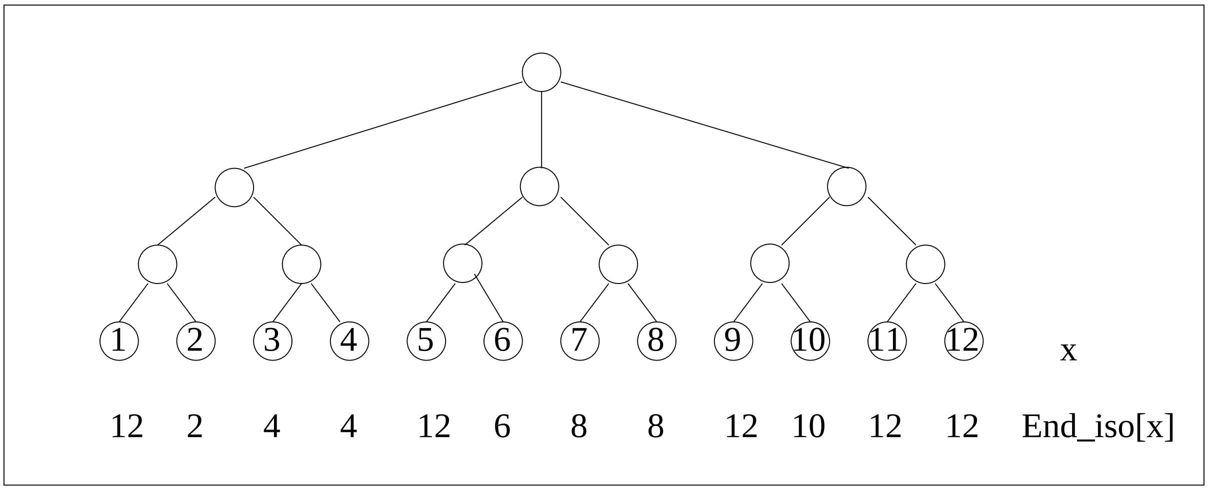

Figure 2: End_iso for the leaf vertices for girth 7

it is critical to describe experiments that can be reproduced with a limited amount of computation so that scientists without access to enormous computing resources will be capable of validating the results. In this regard, our work improves over the previous approach in being both faster and simpler. Further, the ideas illustrated should prove helpful to researchers creating backtrack algorithms for similar problems.

References

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

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

[3] C. T. Benson. Minimal regular graphs of girths eight and twelve. *Canadian J. of Math.*, 18:1091–1094, 1966.

[4] N. L. Biggs. Cubic graphs with large girth. *Ann. New York Acad. Sci.*, 555:56–62, 1989.

[5] N. L. Biggs and M. J. Hoare. A trivalent graph with 58 vertices and girth 9. *Disc. Math.*, 30:299–301, 1980.

[6] F. T. Boesch. On unreliability polynomials and graph connectivity in reliable network synthesis. *J. Graph Theory*, 10(3):339–352, 1986.

[7] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. North-Holland, New York, 1980.

[8] G. Brinkmann, B. D. McKay, and C. Saager. The smallest cubic graphs of girth 9. *Combinatorics, Probability, and Computing*, 4:317–330, 1995.

[9] M. Capobianco and J. C. Molluzzo. *Examples and counterexamples in graph theory*. North Holland, New York, 1978.

[10] C. J. Colbourn. *The combinatorics of network reliablity*. Oxford University Press, New York, 1987.

[11] P. Erdos and H. Sachs. Regulare graphen gegebener taillenweite mit minimaler knotenzahl. *Wiss. Z. Uni. Halle (Math. Nat.)*, 12:251–257, 1963.

[12] W. F. McGee. A minimal cubic graph of girth 7. *Canad. Math. Bull.*, 3:149–152, 1960.

[13] B. D. Mckay. Autoson— a distributed batch system for UNIX workstation networks (version 1.3). Technical Report TR-CS-96-03, Austral. Nat. Univ., Dept. of Computer Science, 1966.

[14] B. D. McKay. Practical graph isomorphism. *Proc. Tenth Manitoba Conf. Numerical Math. and Computing, Cong. Num.*, 30:45–87, 1981.

[15] J. Nadon. Beyond super-λ: counting 4-edge cutsets in 3-regular graphs. Master's thesis, Department of Computer Science, University of Victoria, Victoria, B.C., 1994.

[16] J. Nadon, W. Myrvold, and C. von Schellwitz. Circulants are the worst. In preparation, 1997.

[17] M. O'Keefe and P. K. Wong. A smallest graph of girth 10 and valency 3. *J. Combinatorial Theory, Ser. B*, 29:91–105, 1980.

[18] W. T. Tutte. A family of cubical graphs. *Proc. Camb. Philos. Soc.*, pages 459–474, 1947.

[19] W. T. Tutte. *Connectivity in graphs*. Univ. Toronto Press, Toronto, 1966.

[20] P. K. Wong. Cages— a survey. *J. Graph Theory*, 6:1–22, 1982.

[21] P. K. Wong. On the smallest graphs of girth 10 and valency 3. *Disc. Math.*, 43:119–124, 1983.

[22] P. K. Wong. A note on the problem of finding a (3,9)-cage. *Int. J. Math. and Math. Sci.*, 8(4):817–820, 1985.