MATHEMATICS OF COMPUTATION, VOLUME 28, NUMBER 127, JULY 1974, PAGES 833-838

# Generating the Nine-Point Graphs

By H. H. Baker, A. K. Dewdney and A. L. Szilard

**Abstract.** A program has been written which recently generated all the (unlabelled) nine-point graphs. Written in MACRO-10 assembly language and run on a 165K PDP-10, it generates the complete set of 274,668 graphs in less than six hours. The algorithm on which this program is based is discussed with an emphasis on coding of graphs and various programming techniques designed to save space and time during execution. The methods developed may have applications in other combinatorial generating problems.

The two classic problem types in combinatorial analysis have been existence and enumeration of combinatorial structures. In the latter type of problem, one attempts to produce a formula giving the number of combinatorial structures obeying certain restrictions. It often happens that the descriptions of such structures are not unique and one must take care in the enumeration that no two descriptions correspond to the same structure. If they do, the two descriptions are called "isomorphic". In recent years, a third type of problem called "generative enumeration" [7], has appeared. Here, instead of a formula, one attempts to supply an actual list or catalogue of these structures using a computer.

The generative enumeration of combinatorial structures is usually a challenging and interesting task. Recent efforts in this direction include the generative enumeration of eight-point geometries by Blackburn, Crapo and Higgs [1], generative enumeration of eighteen-point trees by P. Fraser (University of Waterloo—unpublished), generative enumeration of eight-point graphs by Heap [6]. This list is far from exhaustive.

The production of such catalogues of combinatorial structures have a further, pragmatic raison d'être. Mathematicians may use them to check conjectures in low-order cases. Chemists and physicists sometimes find various discrete structures modelled by combinatorial objects: it becomes of interest to observe which structures so catalogued fail to appear in nature.

**General Description of the Algorithm.** Most generative enumeration programs are conceptually divisible into two parts. In the first part, candidate objects are computed or retrieved from some list and in the second part a list is constructed to which the most recently considered candidate object is added if it fails in an isomorphism test when matched against all the objects already in the list. To this basic process. many efficiencies, on both the algorithmic

Received January 12, 1973.

*AMS (MOS) subject classifications* (1970). Primary 05C30, 68A10.

*Key words and phrases.* Graph, generative enumeration, graph isomorphism, filter, preliminary code, filtered code, maximal code.

and programming levels, may be introduced. Our program is first discussed in a general way on the algorithmic level. Shown in Fig. 1 below is a table [3, p. 214] displaying the number of nine-point graphs with from 1 to 18 lines.

| *number of lines* | *number of nine-point graphs* | *lines* | *graphs* | *lines* | *graphs* |
|---|---|---|---|---|---|
| 1 | 1 | 7 | 148 | 13 | 10120 |
| 2 | 2 | 8 | 345 | 14 | 15615 |
| 3 | 5 | 9 | 771 | 15 | 21933 |
| 4 | 11 | 10 | 1637 | 16 | 27987 |
| 5 | 25 | 11 | 3252 | 17 | 32403 |
| 6 | 63 | 12 | 5995 | 18 | 34040 |

FIGURE 1. *Enumeration by numbers of lines of the nine-point graphs*

To generatively enumerate the nine-point graphs with $n \leqq 18$ lines, we catalogue first the nine-point graphs with $n = 1$ lines. A catalogue of nine-point graphs on $n$ lines is produced by having at our disposal a list $G_1^{n-1}$, $G_2^{n-1}, \cdots, G_{f(n-1)}^{n-1}$ of all $f(n-1)$* nine-point graphs on $n-1$ lines. To each graph $G_i^{n-1}$, a line is added to produce a *candidate graph* $G_{ij}^n$. Since there are $36-(n-1)$ ways of adding a new line to $G_{ij}^n$, this many candidate graphs are produced from each $(n-1)$-line graph and $j = 1, 2, \cdots, 36-(n-1)$.

A candidate graph $G_{ij}^n$ is matched against each graph in a list $G_1^n, G_2^n, \cdots, G_l^n$ of graphs on $n$ lines already produced by the algorithm.

If $G_{ij}^n$ is found to be isomorphic with any of the graphs $G_k^n$, it is discarded and the next candidate graph is tried. Many kinds of isomorphism tests may be considered at this point. Although general isomorphism algorithms have received some study, [2] for example, their generality makes them less efficient to implement in low-order cases than "custom-designed" algorithms which take advantage of the small size of the graphs to be tested. In our program, the obvious and most inefficient isomorphism algorithm is used as a basis for the isomorphism test, namely that of permuting the points of $G_{ij}^n$ in all possible ways, checking each time to see whether it has become identical with $G_k^n$. The obvious inefficiency of 9! operations is vastly reduced by interposing various "filters" into this process so that not all 9! permutations need be attempted. A *filter* is an integer-valued function on the points of a graph with the property of being invariant under graph isomorphism. Thus, filters enable us to restrict our permutations to points having the same set of filter values.

Even with such filters present, the number of comparisons to be made can be quite large. The computation corresponding to this comparison must be

*The number $f(n)$ of nine-point graphs having $n$ lines may be computed using Pólya's formula [3, p. 185].

carried out in the order of $N^2$ times, where $N$ is the number of graphs to be generated. For this reason, the algorithm does not carry out these comparisons directly with the graphs $G_k^n$ but generates for each candidate graph $G_{ij}^n$ a special code number $C(G_{ij}^n)$. The actual matching process now proceeds at the speed of the computers unit operations. Although the burden of the above computation (using the filtering process) has been shifted to the production of the code, it must now be carried out in the order of only $N$ times. Of course, even $N^2$ numerical comparisons can be time-consuming. By use of scatter-storage techniques, the number of comparisons was drastically reduced to an average of roughly 4.

**Machine Representation of Nine-Point Graphs.** The nine-point graph generating program was written for a time-shared 165K PDP-10 computer. With a little more than 100K of 36-bit storage words available, and more than 34,000 graphs to store in the biggest case when $n = 18$, it was essential to develop a compact representation for nine-point graphs. In Fig. 2(a) below, a nine-point graph is shown. In Figs. 2(b) and 2(c), the upper half of its adjacency matrix (reflected about the vertical) and its machine representation appear.

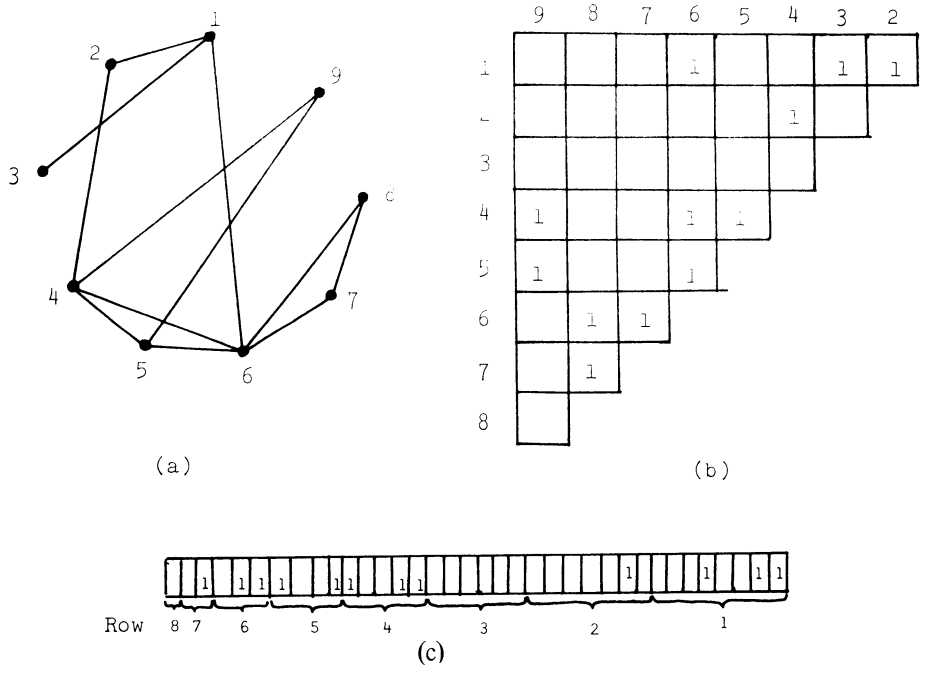


FIGURE 2. *A nine-point graph, its adjacency matrix and* PDP-10 *machine-word representation*

Although only half of the graph's adjacency matrix [3, p. 150] is shown above, its symmetry about the diagonal has been exploited. Thus, the $ij$th

entry of this upper half-matrix is a 1 or a 0 according as the line joining the $i$th and $j$th points is present or absent. As luck would have it, the rows of this upper half-matrix can be strung together into a single 36-bit machine word as shown in Fig. 2(c). One could employ the corresponding representation for a ten-point graph (with 45 possible lines) only by having available a 48-bit (or larger) machine.

**Filters and Codes.** With any graph $G$ on $p$ points, it is possible to associate two sequences of $p$ nonnegative integers. The 1-*degree* (2-*degree*) of a point of $G$ consists of the number of lines (triangles) with which that point is incident. These numbers are arranged in nondecreasing fashion to yield the 1-*degree* (2-*degree*) *sequence*. In the graph shown above, the 1- and 2-degree sequences are 122223345 and 000111222, respectively.

In the generative enumeration process, a graph $G_i^{n-1}$ on $(n-1)$ lines is retrieved from the (previously-generated) list of such graphs. It is retrieved with the above representation, and a given 0-bit is changed to a 1-bit resulting in a *preliminary code* $PC(G_{ij}^n)$ of the corresponding candidate graph $G_{ij}^n$. The 1-degree and 2-degree sequences are now computed for $G_{ij}^n$ from $PC(G_{ij}^n)$ by simple bit-count procedures employing logical operations between various mask words and $PC(G_{ij}^n)$. Corresponding to each point of $G_{ij}^n$, a 2-digit number, consisting of the 1-degree and 2-degree of that point, is created. These numbers are then sorted into nondecreasing order, and the *filtered code* $FC(G_{ij}^n)$ is computed from the preliminary code as the 36-bit word corresponding to an adjacency matrix for $G_{ij}$ whose rows and columns are consistent with this order. Continuing with the above example, the 2-digit numbers corresponding to the points labeled 1 through 9 are 30, 20, 10, 42, 32, 52, 21, 21, 21. These numbers are given a nondecreasing order: 10, 20, 21, 21, 21, 30, 32, 42, 52. One permutation of the points of this graph consistent with this order is 329871546. There are 3! such permutations in this case, corresponding to each of the ways one may permute the points whose filter-value is 21. Originally, the machine representation of this graph (interpreted as an octal number) was 134630001023. The representation corresponding to the above permutation is 744213024020. Both the preliminary and filtered codes of course represent $G_{ij}^n$ and no other graph. The filtered code is much closer to being a unique representation of this graph since there are (in general) far fewer filtered codes for $G_{ij}^n$ than preliminary codes. By use of scatter-storage techniques [5], it is more quickly checked whether the filtered code is already stored in the list of graphs $G_k^n$ than to further improve the code in some manner. This is particularly of advantage when such a code is found already in the list. The candidate graph $G_{ij}^n$ can then be rejected. Now, the new candidate graph $G_{i,j+1}^n$ is generated unless $j=36-(n-1)$, in which case the new candidate graph is $G_{i+1,1}^n$.

If such a code is not found on the list, new codes are computed from the filtered code. These codes are the new machine words corresponding to the

results of orderly permutations of the points of $G_{ij}^n$ having the same filter-value. As soon as a code having a two's complement binary value larger than $FC(G_{ij}^n)$ results from this process, the program hashes to the corresponding location. If an identical code is found there, $G_{ij}^n$ is rejected (as above). If no such code is found, more codes are generated corresponding to further permutations of the points of $G_{ij}^n$. Sooner or later, either a matching code is found in memory or the available permutations are exhausted. In this case, $G_{ij}^n$ is added to the list in the form of its most recent code $MC(G_{ij}^n)$, called the *maximal code* of $G_{ij}^n$. The maximal code uniquely represents $G_{ij}^n$ relative to the algorithm and the list $G_1^{n-1}, G_2^{n-1}, \cdots, G_{f(n-1)}^{n-1}$ of graphs on $n-1$ lines. The method of generating the maximal code is based on Johnson's "adjacent-mark" method of generating permutations [4]. This method was found to be most efficient for our purposes.

**Further Efficiencies.** After the final run was made, another possible filter was discovered. It would appear that a major efficiency may be introduced by checking for similar points [3, p. 171] in a candidate graph. Two such points are adjacent to the same set of points in the graph. A quick survey of exemplar graphs indicated that two points with the same 1- and 2-degree were not very likely to be similar. If two such points were not similar, then all code productions corresponding to any permutation interchanging that pair of points could be bypassed. A similarity test between a pair of points may be performed more quickly than a 2-degree computation for a single point. Of course, there was no opportunity to implement this technique in time for the run. However, it is an interesting problem to analyse the savings yielded by various filters as against the cost of introducing them.

The nine-point graphs on from 1 to 18 lines have been stored on 9-track magnetic tape and are available (with instruction)** from the authors at a tape cost of $20.00. All nine-point graphs having more than 18 lines may be easily generated by the user of this tape by taking the 1's complement of every graph on it with up to 17 lines.

The authors would like to thank Mr. George Lake, director of the University of Western Ontario Computing Centre, for his valuable assistance in making available special conditions for many of our runs. We would also like to thank the University of Western Ontario Research Council for the computing grant which made the resulting catalogue possible.

Computer Science Department
University of Western Ontario
London, Ontario, Canada

1. J. F. BLACKBURN, H. CRAPO & D. A. HIGGS, "A catalogue of combinatorial geometries," *Math. Comp.*, v. 27, 1973, pp. 155-166.

2. D. G. CORNEIL & C. C. GOTLIEB, "An efficient algorithm for graph isomorphism," *J. Assoc. Comput. Mach.*, v. 17, 1970, pp. 51-64. MR 43 #4703.

**The tape must be reformatted for IBM machines.

3. F. HARARY, *Graph Theory*, Addison-Wesley, Reading, Mass., 1969. MR **41** # 1566.

4. S. M. JOHNSON, "Generation of permutations by adjacent transposition," RM-3177-PR, The RAND Corp., Santa Monica, Calif., 1962; *Math Comp.*, v. 17, 1963, pp. 282-285. MR **28** # 2980.

5. R. MORRIS, "Scatter storage techniques," *Comm. ACM*, v. 11, 1968, pp. 38-44.

6. R. C. READ, *Graph Theory and Computing*, Academic Press, New York, 1972.

7. M. B. WELLS, *Elements of Combinatorial Computing*, Pergamon Press, Oxford, 1971. MR **43** # 2819.