Enumerate all non-isomorphic graphs of a certain size

Asked 10 years, 3 months ago Modified 4 years, 5 months ago Viewed 9k times

35

I'd like to enumerate all undirected graphs of size n, but I only need one instance of each isomorphism class. In other words, I want to enumerate all non-isomorphic (undirected) graphs on n vertices. How can I do this?

More precisely, I want an algorithm that will generate a sequence of undirected graphs $G_1, G_2, \ldots, G_k$, with the following property: for every undirected graph G on n vertices, there exists an index i such that G is isomorphic to G_i. I would like the algorithm to be as efficient as possible; in other words, the metric I care about is the running time to generate and iterate through this list of graphs. A secondary goal is that it would be nice if the algorithm is not too complex to implement.

Notice that I need to have at least one graph from each isomorphism class, but it's OK if the algorithm produces more than one instance. In particular, it's OK if the output sequence includes two isomorphic graphs, if this helps make it easier to find such an algorithm or enables more efficient algorithms, as long as it covers all possible graphs.

My application is as follows: I have a program that I want to test on all graphs of size n. I know that if two graphs are isomorphic, my program will behave the same on both (it will either be correct on both, or incorrect on both), so it suffices to enumerate at least one representative from each isomorphism class, and then test the program on those inputs. In my application, n is fairly small.

Some candidate algorithms I have considered:

- I could enumerate all possible adjacency matrices, i.e., all symmetric $n \times n$ 0-or-1 matrices that have all 0's on the diagonals. However, this requires enumerating $2^{n(n-1)/2}$ matrices. Many of those matrices will represent isomorphic graphs, so this seems like it is wasting a lot of effort.
- I could enumerate all possible adjacency matrices, and for each, test whether it is isomorphic to any of the graphs I've previously output; if it is not isomorphic to anything output before, output it. This would greatly shorten the output list, but it still requires at least $2^{n(n-1)/2}$ steps of computation (even if we assume the graph isomorphism check is super-fast), so it's not much better by my metric.
- It's possible to enumerate a subset of adjacency matrices. In particular, if G is a graph on n vertices $V = \{v_1, \ldots, v_n\}$, without loss of generality I can assume that the vertices are arranged so that $\deg v_1 \le \deg v_2 \le \cdots \le \deg v_n$. In other words, every graph is isomorphic to one where the vertices are arranged in order of non-decreasing degree. So, it suffices to enumerate only the adjacency matrices that have this property. I don't know exactly how many such adjacency matrices there are, but it is many fewer than $2^{n(n-1)/2}$, and they can be enumerated with much fewer than $2^{n(n-1)/2}$ steps of computation. However, this still leaves a lot of redundancy: many isomorphism classes will still be covered many times, so I doubt this is optimal.

Can we do better? If I understand correctly, there are approximately $2^{n(n-1)/2}/n!$ equivalence classes of non-isomorphic graphs. Can we find an algorithm whose running time is better than the above algorithms? How close can we get to the $\sim 2^{n(n-1)/2}/n!$ lower bound? I care primarily about tractability for small n (say, $n = 5$ or $n = 8$ or so; small enough that one could plausibly run such an algorithm to completion), not so much about the asymptotics for large n.

Related: Constructing inequivalent binary matrices (though unfortunately that one does not seem to have received a valid answer).

algorithms graphs data-compression graph-isomorphism

Share Cite Improve this question Follow

edited Apr 13, 2017 at 12:48 Community Bot 1

asked Aug 31, 2014 at 21:16 D.W. ♦ 166k 21 230 490

1 Afaik, even the number of graphs of size n up to isomorphism is unknown, so I think it's unlikely that there's a (non-brute-force) algorithm. Regarding your candidate algorithms, keep in mind that we don't know a polynomial-time algorithm for checking graph isomorphism (afaik), so any algorithm that is supposed to run in $O(|\text{output}|)$ should avoid having to check for isomorphism (often/dumbly). (Also, $|\text{output}| = \Omega(n \cdot |\text{classes}|)$.) – Raphael Sep 1, 2014 at 10:35

@Raphael, (1) I know we don't know the exact number of graphs of size n up to isomorphism, but this problem does not necessarily require knowing that (e.g., because of the fact I am OK with repetitions). I don't know why that would imply it is unlikely there is a better algorithm than one I gave. (2) Yes, I know there is no known polynomial-time algorithm for graph isomorphism, but we'll be talking about values of n like $n = 6$ here, so existing algorithms will probably be fast -- and anyway, I only mentioned that candidate algorithm to reject it, so it's moot anyway. – D.W. ♦ Sep 1, 2014 at 22:47

For n at most 6, I believe that after having chosen the number of vertices and the number of edges, and ordered the vertex labels non-decreasingly by degree as you suggest, then there will be very few possible isomorphism classes. At this point it might become feasible to sort the remaining cases by a brute-force isomorphism check using eg NAUTY or BLISS. – Simon Nov 27, 2016 at 10:20

Have you eventually implemented something? This can actually be quite useful. – LeafGlowPath Dec 24, 2019 at 18:02

I've spent time on this. Here is some code github.com/hydroo/ramsey-number-5/blob/... that hopefully gives you an idea. One thing to do is to use unique simple graphs of size n-1 as a starting point. Enumerating all adjacency matrices from the get-go is way too costly. Another thing is that isomorphic graphs have to have the same number of nodes per degree. I.e. "degree histograms" between potentially isomorphic graphs have to be equal. And more infos to be found in the code/comments/readme. out of characters here. – Ronny Brendel Jan 30, 2020 at 14:57

5 Answers

Sorted by: Highest score (default)

25

Probably the easiest way to enumerate all non-isomorphic graphs for small vertex counts is to download them from Brendan McKay's collection. The enumeration algorithm is described in paper of McKay's [1] and works by extending non-isomorphs of size n-1 in all possible ways and checking to see if the new vertex was canonical. It's implemented as `geng` in McKay's graph isomorphism checker `nauty`.

[1]: B. D. McKay, Applications of a technique for labelled enumeration, Congressus Numerantium, 40 (1983) 207-221.

Share Cite Improve this answer Follow

answered Sep 3, 2014 at 3:11

David Eisenstat
984 5 12

I have a problem. I am taking a graph of size `n-1` and extend it by a vertex in all possible ways, as you said. Then I check if the vertex has canonical label, say `1` (canonical vertex?!). However, what if the graph is only isomorphic to the canonical form and the vertex has a different label? I have tried to check the automorphisms to see if the vertex with label `1` is in the same orbit, but then I end up with the graph twice in my list. The paper doesn't really help me. Also, the source code of geng is unreadable due to all those binary optimizations and barely any comments. – Alex Sep 14, 2015 at 14:31

1 @Alex You definitely want the version of the check that determines whether the new vertex is in the same orbit as 1. Could you give an example where this produces two isomorphic graphs? Maybe this would be better as a new question. – David Eisenstat Sep 14, 2015 at 14:43

Okay thank you very much! I guess in that case "extending in all possible ways" needs to somehow consider automorphisms of the graph with `n-1` vertices? e.g. n=3 and my previous graph was P2. Then the two cases where I join the third vertex to one of the previous vertices will of course result in the same graph P3. Could you quickly explain how to properly "extend in all possible ways" or should I ask this as another question? (Also, sometimes I'm confused about what happens, as my program needs to find non-isomorphisms of a special type of graph, which makes things a bit more complicated) – Alex Sep 14, 2015 at 14:57

@Alex Yeah, it seems that the extension itself needs to be canonical. Probably worth a new question, since I don't remember how this works off the top of my head. – David Eisenstat Sep 14, 2015 at 15:04

1 Nauty home page – Guy Coder Dec 30, 2018 at 15:28

5

These papers might be of interest.

"On the succinct representation of graphs", Gyorgy Turan, Discrete Applied Mathematics, Volume 8, Issue 3, July 1984, pp. 289-294 http://www.sciencedirect.com/science/article/pii/0166218X84901264

"Succinct representation of general unlabelled graphs", Moni Naor, Discrete Applied Mathematics, Volume 28, Issue 3, September 1990, pp. 303-307 http://www.sciencedirect.com/science/article/pii/0166218X9090011Z

They present encoding and decoding functions for encoding a vertex-labelled graph so that two such graphs map to the same codeword if and only if one results from permuting the vertex labels of the other.

Moreover it is proved that the encoding and decoding functions are efficient.

The first paper deals with planar graphs. In the second paper, the planarity restriction is removed.

EDIT: This paper might also be relevant:

Graph Isomorphism in Quasi-Polynomial Time, Laszlo Babai, University of Chicago, Preprint on arXiv, Dec. 9th 2015 http://arxiv.org/pdf/1512.03547v1.pdf

Babai's announcement of his result made the news: https://www.sciencenews.org/article/new-algorithm-cracks-graph-problem

But perhaps I am mistaken to conflate the OPs question with these three papers ? The OP wishes to enumerate non-isomorphic graphs, but it may still be helpful to have efficient methods for determining when two graphs ARE isomorphic ?

Share "Cite" Improve this answer edited Apr 17, 2016 at 13:56 answered Apr 12, 2016 at 17:58

Follow

Simon
278 2 6

I appreciate the thought, but I'm afraid I'm not asking how to determine whether two graphs are isomorphic. I really am asking how to enumerate non-isomorphic graphs. Describing algorithms for testing whether two graphs are isomorphic doesn't really help me, I'm afraid -- thanks for trying, though! – D.W. ♦ Apr 18, 2016 at 4:56

Turan and Naor (in the papers I mention above) construct functions of the type you describe, i.e. which map a graph into a canonical representative of the equivalence class to which that graph belongs. If you could enumerate those canonical representatives, then it seems that would solve your problem. – Simon Apr 18, 2016 at 7:17

Babai retracted the claim of quasipolynomial runtime. Apparently there was an error in the analysis. – Raphael Jan 4, 2017 at 21:46

The articles you cited claim linear time for encoding and decoding. If it was possible to use this to test isomorphism, that would make the isomorphism test linear time – agemO Feb 21, 2022 at 20:46

1 They definitely claim that the encoding / decoding is linear time, but they never explicitely claim that the encoding is canonical. In fact reading it again: "If C were a power of 2, this would have been true, since each unlabeled graph has a unique representation in this case." So the representation could have been unique, but it is not the case. – agemO Feb 22, 2022 at 8:26

4

I propose an improvement on your third idea: Fill the adjacency matrix row by row, keeping track of vertices that are equivalent regarding their degree and adjacency to previously filled vertices. So initially the equivalence classes will consist of all nodes with the same degree. When a newly filled vertex is adjacent to only some of the equivalent nodes, any choice leads to representants from the same isomrphism classes. So we only consider the assignment, where the currently filled vertex is adjacent to the equivalent vertices with the highest number (and split the equivalence class into two for the remaining process).

I think (but have not tried to prove) that this approach covers all isomorphisms for $n < 6$. For larger graphs, we may get isomorphisms based on the fact that in a subgraph with edges $(1, 2)$ and $(3, 4)$ (and no others), we have two equivalent groups of vertices, but that isn't tracked by the approach. (It could of course be extended, but I doubt that it is worth the effort, if you're only aiming for $n = 6$.)

A naive implementation of this algorithm will run into dead ends, where it turns out that the adjacency matrix can't be filled according to the given set of degrees and previous assignments. It may be worth some effort to detect/filter these early. Some ideas:

- If the sum of degrees is odd, they will never form a graph
- Fill entries for vertices that need to be connected to all/none of the remaing vertices immediately.

Share Cite Improve this answer Follow

answered Sep 2, 2014 at 13:58

FrankW

6,609 4 26 42

4

There is a paper from the early nineties dealing with exactly this question:

Efficient algorithms for listing unlabeled graphs by Leslie Goldberg.

The approach guarantees that exactly one representant of each isomorphism class is enumerated and that there is only polynomial delay between the generation of two subsequent graphs.

The methods proposed here do not allow such delay guarantees: There might be exponentially many (in n) adjacency matrices that are enumerated and found to be isomorphic to some previously enumerated graph before a novel isomorphism class is discovered.

Share Cite Improve this answer Follow

answered Oct 3, 2017 at 14:00

Pascal Welke
41 1

1

You also may see the paper

A new formula for the generating function of the numbers of simple graphs, Comptes rendus de l'Acade'mie bulgare des Sciences, Vol 69, No3, pp.259-268

http://www.proceedings.bas.bg/cgi-bin/mitko/0DOC_abs.pl?2016_3_02

Share Cite Improve this answer Follow

answered Jul 4, 2020 at 18:17

Leox
163 5