

# Re: Need Graph Isomorphism Algorithm De-bunked

---

*Source:* <http://www.derkeiler.com/Newsgroups/sci.crypt/2006-09/msg00996.html>

---

- *From:* Mike Amling <nospam@xxxxxxxxxx>
  - *Date:* 27 Sep 2006 22:36:45 EDT
- 

fgrieu wrote:

Mike Amling wrote:

François Grieu wrote:

On the positive side, **\*\*IF\*\*** the algorithm works save for an accidental hash collision, I think we can get rid of any possibility of collision, and still be in polynomial time (as Mike conjectured then retracted).  
I (snipped)

Last night I thought I understood what you are suggesting, and if the algorithm works, it would put graph isomorphism into P, not just RP. This morning I'm doubtful.

First, do I understand the what you suggest?

I did not quite recognize my idea. I'll paraphrase you, at the same time fixing a flaw in my earlier description. The algorithm is stated for an undirected graph only (symmetric matrix), no loopy vertices (diagonal is 0).

0. Associated to each node we need
  - a "value" in the range  $0..n-1$ , assimilated to a fixed length bit string of  $\text{ceil}(\lg 2(n))$  bits;
  - a "longhash" bit string;
  - a "result" bit string, initially empty.
1. Repeat 2..5 with each node the special node in turn
2. Assign the special node the value 1. Assign all the other nodes the value 0.
- Repeat 3..4 some number of times ( $N? N/4?$ ).
3. For each node, assign to longhash its value, followed by its neighbors values in sorted order (duplicates kept); longhash has length at most  $n * \text{ceil}(\lg 2(n))$ .
4. Make a list of all the longhash assigned in step 3. Sort the list and remove duplicates (regard a shorter

## Re: Need Graph Isomorphism Algorithm De-bunked

string as strictly less than any longer string).

Assign each node a new value, which is the position of its longhash in the list.

Append to the special node's result a bit string reversibly coding the list (e.g. prefix each longhash with a length prefix of appropriate fixed size, the zero length coding end of list)

Notice that what we just appended is invariant under graph isomorphism, and that it is enough to rebuild the current longhash of each node from its current value.

5. Append to the special node's result the sorted list of the values of each node in sorted order (duplicates kept);

6. Sort all N results (duplicates kept, shorter strings first), compute the whole graph's hash as a bit string reversibly coding the N results (e.g. prefix each result with a length prefix of appropriate fixed size). Notice this graph's hash is invariant under graph isomorphism, and enough to backtrack (except for a shuffle of the nodes) the values and longhash assigned to each node, without knowledge of the graph.

I've coded up something I think is equivalent to what you suggest. It works for up to 7 nodes (simply connected, undirected, no loopback), but for 8 it only found 10126 equivalence classes, not 11117. I'll look again at the code to see if something obvious is amiss. I instrumented it to note the maximum number of iterations needed before the nodes' values duplicate previous values, and it's surprisingly few:

Sufficient

Nodes Iterations

3 1

4 1

5 1

6 2

7 3

8 5

E.g., the fourth iteration of a 7-node graph always generates values that duplicate a previous generation (the immediately previous). For 8 nodes the duplicate may be two generations back.

—Mike Amling

.