An heuristic for graph isomorphisms

Nguy˜ên Lê Thành Dũng Blanchard Nicolas Koliaza

December 8, 2012

Foreword

The problem considered is the graph isomorphism (GI) problem : given two graphs G and G', can one compute an isomorphism between them. The original goal was to implement the Weisfeiler-Lehman heuristic in the C programming language, but we chose to implement our own heuristic to test its performance. The precise goal was to find an heuristic which would behave polynomially on the a bigger subset of problems, even if the polynomial degree increases.

1 Algorithmical Considerations

We adopt the notations of graphs with n vertices and m edges. $G(n,p)$ denotes the graph generated through the Erdös-Rényi model with n vertices and probability p for each edge.

There is a strong link between GI and algebra : finding an isomorphism is the same as finding a basis in which the adjacency matrix of G becomes that of G'. Testing all solutions is equivalent to testing all permutations and a naïve algorithm would take $n!$ operations. The GI problem lies between the complexity classes P and NP and Schöning [4] proved that it is not NP-complete unless $P = NP$. However, no polynomial solution has been found and the best deterministic algorithm so far is due to Eugene Luks and runs in $2^{O(\sqrt{nlogn})}$[1]. However some efficient heuristics are available, including *nauty* and *conauto*, which runs in $O(n^5)$ with very high probability for any graph in $G(n,p)$[2]. In our algorithm and test we often consider multi-graphs, as the hard cases are much easier to generate for multi-graphs and the difference has no impact on our heuristic. Finally, we tried to make an heuristic which would behave well when launched on p different graphs to check if any two are isomorphic.

2 The Heuristics

2.1 The Weisfeiler-Lehman heuristic

The original Weisfeiler-Lehman (WL) heuristic works by coloring the edges of a graph according to the following rules :

- We begin with a coloring that assigns to every vertex the same color (this is the 1-dimensional version).
- At each pass, the color of each vertex is determined by the number of neighbours of color c for each c.

- After at most n passes, the colors don't change anymore.

These rules actually only produce a certificate of non-isomorphism. To construct an isomorphism using WL one uses backtracking coupled with the fact that the image of a vertex has to be of the same color. It is easy to see that two isomorphic graphs behave in the same way when subject to WL coloring, but the converse does not generally hold. Notably, some special classes of graphs make backtracking omnipresent, e.g. in k-regular graphs, thus leading to time complexity up to $O(n^n)$. The heuristic can be improved by changing the initial coloring of the vertices, but we shall only study this case.

2.2 Another heuristic

2.2.1 The idea

Our heuristic is based on the following property :

Let $V_k(x)$ be the number of neighbours at distance exactly k from x, and $P_k(x)$ the number of paths of length k starting from x, then if f is an isomorphism between G and G' , $V_k(x) = V_k(f(x))$ and $P_k(x) = P_k(f(x))$. Thus, by computing the different V_x and P_x we can prune the search tree and limit the possibilities. We name the array of couples $P_k(x), V_k(x)$ for k between 1 and n PV(x), and compute an array containing PV(x) for each x, obtaining the $n \times n$ matrix PV.

2.2.2 The algorithm

The algorithm we use actually incorporates multiple testing phases to quickly eliminate easy cases. It can be decomposed in the following steps :

1. Lecture and choice of data structure
2. Primary test phase
3. Construction of each PV-array
4. Sorting of each PV-array
5. Comparison of the PV-arrays
6. If possible construction of an isomorphism using backtracking

2.2.3 First test phase

The problem with the naïve algorithm is that the number of possibilities groes exponentially in n, however, when one considerons permutations which are composed of one transposition, there are at most n^2possibilities and each take at most m operations. The first test phase uses different simple techniques to quickly find a certificate of non-isomorphism or an easy isomorphism when doing so is possible. The tests are run in this order :

1. Check for equality between the matrices (runs in $O(m)$)
2. Check size and number of connected components (in $O(m)$)
3. Compare the list of degrees (in $O(m + n * log(n))$) and record the number of possibilities.

4. When the number of possibilities is small ($O(n^2)$) use bruteforce (in $O(n^2 * m)$).

5. Still check for simple transpositions when it is not the case (also in $O(n^2 * m)$)

2.2.4 Details and optimization

The choice of data structures varies between adjacency matrix and list, and sometimes both, and allows us to take advantage of faster algorithms on sparse graphs using lists, while allowing us to know in O(1) if two vertices are linked.

The construction of the V-array is generally the most time-expensive task, it is mostly done by multiplication of the adjacency matrix (or by equivalent operations on lists). Sorting the V-array is easily done with a heapsort. The construction of the isomorphism is done by trying to assign to an x an y such that$V(x) = V(y)$ and backtracking when unsuccessfull.

Some code optimizations have also been added : the construction of V-arrays by matrix multiplication can be parallelized easily, and multi-threading has been used to do so. We also used it for sorting the arrays. However we didn't use Strassen's algorithm because for the matrix size considered ($n \leq 500$) the gain is very small and mostly compensated by the overhead.

3 Tests and Benchmark

We have included different basic test generation programs, depending on the model :

3.1 Random generation

- The Erdös-Rényi $G(n, p)$ model with probability p for each edge works in general quite fast because the probability that two graphs are not isomorphic (and trigger a certificate) is very high.
- The similar model with $G(n, M)$ behaves the same way.
- A model to generate random k-regular multigraphs that generates k random permutations of n vertices and links each vertice of the initial array to each of its images in the resulting permutations.

All other methods so far for generating k-regular graphs (and not multigraphs) are probabilistic except for very small k, and when $k \geq log(n)$ have exponential expected time [3], so we restricted ourselves to those three models, as our algorithm should work as well on multigraphs as on normal graphs.

3.2 Isomorphic graph generation

To generate two isomorphic graphs is actually really easy : as two graphs are isomorphic if and only if their adjacency matrices are similar, one has to compute a random permutation (which corresponds to a change of basis), and to apply it to any graph to obtain an isomorphic copy. We generate those from all three different models of random graphs already implemented.

3.3 Benchmark

4 Comparison with Weisfeiler-Lehman

Our goal here is to show that the subset of problems solved in polynomial time by WL is strictly included in the subset solved by our heuristic. We shall consider an extended version of WL to facilitate the proof, where colors aren't redistributed among $[1;n]$ but are instead an injective function of the vertex's previous color and the multiset of its neighbour's colors. When WL behaves polynomially, it does so by either finding that two vertices must be image of one another by the isomorphism (if it exists) or by finding a negative certificate. We must show those three properties :

- When WL colors a vertex in an unique color in both graphs, then extended WL does the same, and our algorithm also recognizes that those two vertices must be linked if the graphs are isomorphic (polynomial search of isomorphism)
- When WL finds two incompatible coloring, then so does extended WL, and our algorithm also gives a negative certificate (polynomial negative certificate)
- When the graphs are isomorphic, WL, extended WL and our heuristic compute the isomorphism (correctness)

The third is immediate, because the operations we use are invariant through permutations of the vertices. The two other properties hold for extended WL, and we must show that it implies that they hold for our algorithm.

Let's prove the first one by supposing that at one point extended WL colors exactly one vertex in each graph in the same color. At each step the color is uniquely determined by the previous color and its neighbours' colors. We procede by induction on the number of steps : at step 1, the color is determined by the degree of the vertex so our algorithm also manages to find that it is the only two vertices with that same degree. Now let us suppose that the information our algorithm gathers is greater than the information extended WL has at step k. Then at step k+1, we know the previous color of a vertex and the previous color of all its neighbours, so

Let us suppose one finds a coloring using WL that produces a certificate of non-isomorphism. It does so at the k^{th}iteration. If $k = 1$ then it is simply that the degree list that mattered and our heuristic also managed to notice it (it does it quickly thanks to the primary test phase, but the PV-array would suffice). Now suppose that the two colorings are incompatible, it means that the number of vertex of one color isn't the same in both graphs. That means that there are two vertices such that they were identically colored at $k-1$ but that their neighbourhood were not identically colored.

References

[1] László Babai and Eugene M. Luks. Canonical labeling of graphs. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, STOC '83, pages 171–183, New York, NY, USA, 1983. ACM. 1

[2] José Luis López-Presa and Antonio Fernández Anta. Fast algorithm for graph isomorphism testing. In *SEA*, pages 221–232, 2009. 1

[3] Brendan D. McKay and Nicholas C. Wormald. Uniform generation of random regular graphs of moderate degree. *Journal of Algorithms*, 11(1):52–67, March 1990. 3.1

[4] Uwe Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37:312–323, 1988. 1