An heuristic for graph isomorphisms

Nguyễn Lê Thành Dũng Blanchard Nicolas Koliaza

December 1, 2012

1 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.

2 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.

The GI problem lies between the complexity classes P and NP and Schöning [3] 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 $2^{O(\sqrt{n \, log n})}$[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 heuristic work the same.

3 The Heuristics

3.1 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.
- 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. On G(n,p) graphs, one can show that the probability of two graphs being non-isomorphic while admitting the same coloring tends towards 0 as n increases. However some special classes of graphs do not behave in such a way, and backtracking becomes omnipresent, notably in k-regular graphs, thus leading to time complexity up to $O(n^n)$.

3.2 Another heuristic

3.2.1 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, then if f is an isomorphism between G and G' , $V_k(x) = V_k(f(x))$. Thus, by computing the different V_x we can prune the search tree and limit the possibilities. We name the array of degrees $V_k(x)$ for k between 1 and n V(x), and compute an array containing V(x) for each x, obtaining the $n \times n$ matrix V.

3.2.2 The algorithm

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

1. Lecture and choice of data structure
2. Separation into connex components
3. Primary test phase
4. Construction of the V-array
5. Sorting of the V-array
6. Comparison of the V-arrays
7. If possible construction of an isomorphism using backtracking

3.2.3 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 primary test phase relies on the degree list and connex components to check if a quick certificate of non-isomorphism can be found.

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.

4 Benchmarks

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

The Erdös-Rényi 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

5 Comparison with Weisfeiler-Lehman

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.

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

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