An heuristic for graph isomorphisms

Nguyễn Lê Thành Dũng   Blanchard Nicolas Koliaza

December 15, 2012

Foreword

The problem considered is the graph isomorphism (GI) problem : given two graphs G and G', can one compute an isomorphism between them. We implement our own heuristic called PN for Path-Neighbour. This is the short version of the report, and more details can be found in the complete version.

1 The Heuristics

1.1 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. We then make one random choice and use backtracking before coloring again.

Two isomorphic graphs admit the same coloring, but the converse does not always hold: k-regular graphs for example are pathological cases which take exponential time.

1.2 The PN heuristic

1.2.1 The idea

PN is based on the following property :
Let $N_k(x)$ 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' , $N_k(x) = N_k(f(x))$ and $P_k(x) = P_k(f(x))$. Thus, by computing the different N_x and P_x we can prune the search tree and limit the possibilities. We name the array of couples $P_k(x), N_k(x)$ for k between 1 and n PN(x), and compute an array containing PN(x) for each x, obtaining the PN-arrays.

1.2.2 Structure of 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. Input parsing and choice of data structure
2. Primary test phase (a collection of fast tests to quickly eliminate easy cases)
3. Construction and sorting of each PN-array
4. Comparison of the PN-arrays using the neighbours
5. If possible construction of an isomorphism by refined bruteforce

2 Comparison with Weisfeiler-Lehman

2.1 Proof

We can show that PN behaves polynomially on a subset of problems that strictly includes the subset where WL behaves polynomially. We consider an extended version of WL which is very similar and facilitates the analysis (even though it is much less efficient in practice). The proof is done by considering the cases when WL prunes efficiently the search tree and by showing by induction that PN does the same in those cases. The detailed proof is in the long report.

2.2 Complexity analysis

As the GI problem isn't known to be in P, it is not absurd to have a worst running time of $O(n!)$. However the running time is generally lower, and PN often takes $O(n^4)$. With the primary test phase, it can go down to $O(n^2m)$ or even $O(m + n * log(n))$ in some cases. This means that WL (which can run in $O(nm)$) is often more efficient, but PN is safer (the pathological cases are not as frequent).

2.3 Problems, optimization, and improvements

The biggest problem was that the numbers in the PN-arrays quickly grow out of proportions so one has to to multiplication modulo p, which means that our implementation is only probabilistic. PN might be improved by using adjacency lists in the case of sparse graphs, but this would require a lot of new code for an improvement that might not be sizeable. Some other improvements would target optimization of the mulp function, parallelization inside the matrix multiplication and interlacing neighbourhood checks with PN-array generation.