Project Documentation Group 07

Janiek Smulders, Jayanshi Tripathi, Luyao Wang and Ruilin Yang

University of Twente

j.l.smulders@student.utwente.nl, j.tripathi@student.utwente.nl, l.wang-6@student.utwente.nl, r.yang-1@student.utwente.nl

April 19, 2020

According to the following documentation of computational results for the graph isomorphism project, together with the 0.3 bonus we have got, we aim at a grade of 8.5.

1. Category 6: Basic Problem Instances

The basic problem instance have been solved correctly during the online project delivery session. The following table shows the computation times.

instance	correctly solved	comp. time (s)
basicGI1	✓	0.03
basicGI2	✓	0.02
basicGI3	✓	2.66
basicGIAut	✓	3.46
basicAut1	✓	2.18
basicAut2	✓	1.66

Table 1: *Computation times basic instances.*

2. Category +1: Additional Techniques for Faster Algorithms

2.1. Branching Rules

Not implemented.

2.2. Preprocessing

2.2.1 Trees

Not implemented.

2.2.2 Preprocesing using Twins or Modules

We had an idea on the detection and merging of twins. Two vertices in a False Twin class have the same neighboring vertices hence the same coloring in the stable partition. Two vertices in a True Twin class has the same base neighboring vertices (neighboring vertices except the true twins), hence they will also have the same color in the stable partition.

The idea is to represent each twin class with only one vertex, so that the disjoint union of all graphs would have less vertices. By generalizing the vertices into twins classes, it cuts down the calculation complexity of color refinement algorithm, therefore speeds up the GI problems for some of the graphs with multiple classes of twins.

However due to time limit, we didn't finish a correct implementation of it.

3. Category +1: Implementation of Fast Partition Refinement

Based on the algorithmic idea of Hopcroft's algorithm for DFA minimization, and implementation idea of Color Refinement and its Applications(2017)[1], we implemented a fast refinement algorithm. This implementation, although not using Doubly Linked List, has speed up the refinement of graphs with long paths significantly.

			comp. times (s)
instance	instance size $\|V\|$	standard	fast part. ref.
Threepaths320	960	0.78	0.06
Threepaths640	1920	2.97	0.16
Threepaths1280	3840	13.83	0.54
Threepaths2560	7680	62.98	2.31
Threepaths5120	15360	230.36	8.29
Threepaths10240	30720	872.01	34.32

Table 2: *Computation times with and without fast partition refinement.*

In the main program, the use of fast refinement to determing partition of vertices can be enabled by the flag "use_wk5" in both GI_problem and AUT_problem function.

4. Category +1: Using Generating Sets for Computing |Aut(G)|

We have implemented tree pruning and counting number of automorphism by multiplication with a small number of permutations in a generating set, rather than traversing the entire branching tree.

instance	without gen. set	with gen. set
torus24	0.47	0.32
basicGIAut	6.97	3.46
torus72	19.92	16.01
products72	23.89	26.73
torus144	247.28	199.66

Table 3: *Computation times with and without using generating set to prune the branching tree*

5. Category +1: Additional and Genuinely New Ideas

5.1. Theorem

If graphs A and B are isomorphic, graph C is not isomorphic with A, then C is not isomorphic with B.

5.2. Corollary

Here the group is a list of graphs that are deemed as potentially isomorphic by one call of color refinement. Now branching alogrithm is needed to determine whether every two graphs within this group is isomorphic. On natural thought, it may need "n choose 2" calls to the branching algorithm. But based on the idea, we can shrink the number of calls made to the branching algorithms to at most O(n) times. This resulted in a significant speedup in finding the isomorphic graphs from a list. we have therefore used the following algorithm to avoid redundant comparisons:

```
typify_group(group, list_of_graphs):
    types = []
    typified = []
    for i in range(0, len(group)-1):
        if group[i] not in typified:
            matches = []
            for j in range(i+1, len(group)):
                if group[j] not in typified:
                    if is_iso(list_of_graphs, [group[i], group[j]]):
                        matches.append(group[j])
                    else:
                        pass
            typified.append(group[i])
            typified.extend(matches)
            types.append([group[i]] + matches)
    return types
```

The computation time that we have obtained with and without this algorithm typify are given in the following table. As can be seen, for larger problem instance, there is a visible improvement compared to without this typify idea.

instance	without typify	with typify
products72	27.67	21.52
basicGIAut	3.13	2.50
basicAut1	1.72	1.70
basicAut2	1.41	1.41

Table 4: *Computation times with and without using typify algorithm to decrease isomorphic comparisons*

6. Balance Sheet & Reflection

6.1. Work Distribution

The following is the estimated work distribution in the group for this implementation project.

	J. Smulders	J. Tripathi	L. Wang	R. Yang
Color Refinement	–%	–%	25%	75%
Branching Algorithm	–%	–%	–%	100%
Preprocessing Twins	50%	–%	50%	–%
Fast Part. Refinement	–%	–%	–%	100%
Generating Set	–%	20%	20%	60%

Table 5: *Work Distribution Group 07.*

6.2. Team Dynamics

Our positive and/or negative experiences with respect to teamwork and team composition in the project group, specifically with respect to the interdisciplinary team composition are:

+ The team consists of 3 TCS students and 1 AM students. The AM student experienced new things like SCRUM meeting and GitLab. The TCS students observed the highly abstract thinking when the AM student raise the algorithmic idea of processing twins.
+ Since Apr.7, we've had a SCRUM meeting every morning. The attendance is perfect.
- There was an inharmonious argument near the deadline concerning workload distribution. Such issues can be better handled if the relevant people can have conversation in private at first.

References

[1] Martin Grohe, Kristian Kersting, Martin Mladenov, Pascal Schweitzer (2017) Color Refinement and its Applications. Page 4-5.