Project Documentation Group 3

Dario Capitani - s2754194
Gyum Cho - s2113201
Junseo Kim - s2648687

April 18, 2023

Based on the following documentation of our group isomorphism project, we aim for a group-average grade of 9. Inline with Section 5, the individual grades are equal. Our group consisted of 3 members instead of 4, so we decided that the extra point be divided into 0.33 for each member. In addition, 5 out of 10 bonus instances were solved during the delivery session, so an extra 0.5 is added.

Team member	**Grade**
Dario Capitani	9 (+0.33) (+0.5)
Gyum Cho	9 (+0.33) (+0.5)
Junseo Kim	9 (+0.33) (+0.5)

Table 1: *Suggested grades*

1. Category 6: Basic Problem Instances

The basic problem instances have all been solved correctly during the project delivery session. The computation times of the instances are listed below.

Instance	**Correctly solved**	**Comp. time (s)**
Basic1GI.grl	✓	1.53
Basic2GI.grl	✓	0.71
Basic3GI.grl	✓	0.04
Basic4Aut.gr	✓	0.07
Basic5Aut.grl	✓	0.14
Basic6GIAut.grl	✓	0.12

Table 2: *Computation times of basic instances*

2. Category +1: Additional Techniques

2.1. Branching Rules

After the implementation of our branching algorithm, we compared the performances of 3 different branching rules.

- Branching with a vertex from an arbitrary color class
- Branching with a vertex from the color class with minimal length
- Branching with a vertex from the color class with maximal length.

After running multiple experiments on different instances, we saw that branching from the maximal color class proved to be most optimal in almost all cases. Tree structures were problematic and did not follow the trend of resulting in optimal times when branching from the maximal color class. However, because the times for trees were similar for all 3 rules and because the times of all other instances were significantly reduced, we chose to implement branching from the maximal color class for the project.

The results of each of the branching rules for solving the GIAut problem with standard partition refinement are listed below.

	Comp. time (s)		
Instance	**Arbitrary color class**	**Minimal color class**	**Maximal color class**
torus144.grl	36.8	43.67	<u>28.36</u>
products216.grl	140.29	149.31	<u>120.23</u>
trees90.grl*	48.26*	<u>45.75</u>*	50.45*
modulesD.grl	35.73	50.48	<u>30.70</u>
cubes6.grl	74.50	69.31	<u>62.22</u>

* Tree structures did not follow the trend, but the 3 results were similar

Table 3: *Differences in computation times for various branching rules*

2.2. Preprocessing

Not implemented

3. Category +1: Fast Partition Refinement

We implemented fast partition refinement based on Hopcroft's algorithm that was taught in the lectures. For optimization, a doubly linked list was used for storing the color classes, along with a queue for colors left to be considered. The speedup in the refinement process was significant and improved performance in all areas of the project. The computation times for reaching a stable coloring with and without fast partition refinement are listed below.

		Comp. time (s)	
Instance	**Instance size**	**Standard partition refinement**	**Fast partition refinement**
threepaths640.gr	1920	0.69	0.04
threepaths1280.gr	3840	2.78	0.14
threepaths2560.gr	7680	11.37	0.54
threepaths5120.gr	15360	47.32	2.08
threeptahs10240.gr	30720	213.10	8.46

Table 4: *Computation times with and without fast partition refinement*

4. Category +1: Generating Sets for Automorphism Groups

We implemented the algorithm for computing the generating set for automorphism groups based on the lecture. The generating permutations were efficiently computed by pruning the recursion tree (Pruning Lemma). To calculate the order of the generating set of the automorphism, the Orbit-stabilizer Theorem was used. The non-trivial orbit and corresponding stabilizer as a generating set is computed by the provided modules. The order of the orbit is easily computed, and the order of the stabilizer is computed recursively, since it is also a generating set. Their product returns the total number of automorphisms. The computation times for solving the GIAut problem with and without the algorithm are listed below.

	Comp. time (s)	
Instance	**Original branching algorithm**	**Generate automorphism algorithm**
products216.grl	101.57	61.86

cographs1.grl	DNF	4.79
trees90.grl	59.19	1.36
modulesC.grl	DNF	103.35
bigtrees1.grl	DNF	8.20

Table 5: *Computational improvements of using the generate automorphism algorithm*

5. Balance Sheet & Reflection

5.1 Work Distribution

The following section is an estimation of the work distribution among the team members during the project.

	D. Capitani	**G. Cho**	**J. Kim**
Branching Algorithm	33%	33%	33%
Branching Rules	33%	33%	33%
Fast Partition Refinement	33%	33%	33%
Automorphism Groups	33%	33%	33%
Documentation	33%	33%	33%

Table 6: *Work distribution Group 3*

5.2 Team Dynamics

The work distribution was equal and fair because we met physically for all project work. The component to work on would be decided prior to the meeting, and each member prepared for the meeting by going over the lecture material for that component.

During the meetings, we began by brainstorming on how the problem could be solved. If a member had trouble understanding a certain topic, the other members would explain in detail. Once a solution was thought out, pseudocode was first written, then it was coded and documented.

Due to this approach of working together, each member was fully aware of the workings of each component of the project.