Project Documentation Group 28

Jelle Boon, Duc Duc Tran, Minh Vo and Salih Eren Yücetürk

University of Twente

j.p.boon-1@student.utwente.nl, d.d.tran@student.utwente.nl,

vonhatminh@student.utwente.nl, s.e.yuceturk@student.utwente.nl

April 14, 2022

Behind is the grade deviation for our group. We want to honor Minh for his great contribution over our project, especially on solving bonus instances.

Team member	grade (deviation from average)
Jelle Boon	(-0.3)
Duc Duc Tran	(-0.2)
Minh Vo	(+1.0)
Salih Eren Yücetürk	(-0.5)

Table 1: *Suggested grades*

1. Basic Problem Instances

We have been able to solve the basic problem instances, and they have been solved correctly. The computation times of our program are shown below inside table 2

instance	correctly solved	comp. time (s)
basic1GI	✓	1.03
basic2GI	✓	1.33
basic3GI	✓	0.13
basic4Aut	✓	0.001
basic5Aut	✓	0.027
basic6GIAut	✓	0.137

Table 2: *Computation times basic instances.*

2. Additional Techniques for Faster Algorithms

2.1. Branching Rules

We have 2 simple branching rules but it boosts the running time significantly in certain cases:

1: With twins detection, we can have a rule when pairing vertices they need to be the same type of twin(false/true/non-twin) and have the same number of twins that they are mapped to

2: When choosing color class for branching, we choose color class with biggest size. We see that there are a big difference in certain cases

instance	instance size $\|V\|$	comp. times (s) standard without rules	with branching rules
modulesC	30	8.2	14.8
3130bonus05GI	350	51.3	4.9
3130bonus10Aut	86	585	3.6
wheelstar15	65	40.7	14.72
wheelstar16	86	1020	28

Table 3: *Computation times solving GIAut problems without and with branching rules.(using group isomorphism)*

2.2. Preprocessing

2.2.1 Trees

We used a simple BFS for counting connected vertices to recognise trees. The idea of algorithm goes as follows: first find a (double-)center(s) of the tree, then build the tree according to one of its centers as root and finally encode them using "(" and ")".

Two trees are isomorphic if and only if their encodings are equal. We can count the possible mappings between two isomorphic trees by looking at the coding of just one tree. Computation time for recognition is moderate, while the speedup for trees is significant. Therefore this process is really worth while.

instance	instance size $\|V\|$	comp. times (s) standard	incl. tree detection
products216	216	64.3	62.6
3130bonus07Aut	181	4.26	0.006
trees36	36	0.032	0.003
trees90	90	0.13	0.005
bigtrees2	71	0.07	0.002
bigtrees3	227	7.09	0.009

Table 4: *Computation times solving GIAut problems for trees.*

2.2.2 Preprocesing using Twins

We have implemented the pre-processing detection of true and false twins. For each graph, we just have to mark and keep twin masters, compute the number of its mapped twins and ignore those mapped twins. In case of true twins, we just have to find all vertices with same neighbors set. For false twins, by adding the vertice to its own neighbors set we can compare and find them. The result is speedup significanty in counting the number of isomorphisms for most of the graphs because of only processing twin master and ignoring other twins. After counting all isomorphism without redundant twins we just have to multiply it with factorial of number of twins for each twin master. However, because we not delete the other twins, the computation time can be slightly slower in some cases. See the table 5 for sample results on some instances.

instance	instance size $\|V\|$	comp. times (s) without twin detection	comp. times (s) incl. twin detection
3130bonus03GI	1267	28.1	31.4
products72	72	1.7	3.3
wheeljoin14	15	0.11	0.04
wheeljoin25	26	1.3	0.22
wheeljoin33	34	10.59	1.52
modulesC	30	118	9.7
cographs1	22	0.22	0.007

Table 5: *Computation times solving GIAut problems (except bonus03GI) with and without detection of twins (using group isomorphism)*

3. Implementation of Fast Partition Refinement

We have implemented the fast partition refinement Algorithm based on Hopcroft's algorithm for DFA minimization of finite automata. In oder to achieve a speedup with this algorithm, we have to use the set implementation of Python which works similarly to doubly-linked-list due to its unique indexing. The speedup on some sample instances with this algorithm for computing a stable coloring is as follows in table 6.

instance	instance size $\|V\|$	comp. times (s) standard Loading time	standard Run time	fast part. ref. Loading time	fast part. ref. Run time
Threepaths80	80	0.0015	0.027	0.003	0.003
Threepaths1280	1280	0.08	6.25	0.10	0.043
Threepaths10240	10240	7.2	540	7.13	0.23
torus72	72	–	4.83	–	2.00
products72	72	–	3.47	–	1.45
products216	216	–	105.85	–	52.7

Table 6: *Computation times with and without fast partition refinement.*

4. Using Generating Sets for Computing |Aut(G)|

We have implemented a recursive algorithm that is based on the orbit-stabilizer theorem to create a generating set of all permutations of a graph, that was presented to us during the lecture about the Automorphism Groups.

Theorem 4.1. *For any vertex α in a graph (H), the order of that graph is equal to the product of the orders of the stabilizer and orbit around α. In maths: For any $\alpha \in V$, $\quad |H| = |H_\alpha||\alpha^H|$*

This theorem is really useful, it allows us to transform our problem into a multiplication of terms that are relatively easy to calculate as follows: If there are points, which orbit does not solely consist of themselves. If so, pick them as your α in the Orbit-Stabilizer Theorem and repeat this until there are no more points with a non-trivial orbit. Then set your $|H_\alpha|$ to 1 and multiply all the orders of the orbits together.

Counting isomorphism is boosted significantly since we don't have to reach all case of permutation to count according to pruning lemma. If we have found all permutations that follow mapNum(x) (mapping number of x) = mapNum(y) we just have to find 1 more permutations follow mapNum(x) != mapNum(y) and go back to trivial ancestor node which will save us a lot of times

instance	$\|V\|$	comp. times (s) normal counting	group counting
cubes6	64	75.33	0.85
cubes7	128	1258	6.2
cubes8	256	–	41.87
torus144	144	26.62	15.27
products216	216	131.97	54.11
modulesD	144	39.19	14.70

Table 7: *Computation times for trees.*

5. Additional and Genuinely New Ideas

In this project we really focused on getting the ideas, theorems and propositions that were presented to us during the lectures working as fast and efficiently as possible. This resulted in the fact that we did not really work on or look at other ways to tackle the graph isomorphism problems. Therefore we do not have anything else to add to this section.

6. Balance Sheet & Reflection

6.1. Work Distribution

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

	J. Boon	Minh Vo	Duc Duc Tran	S. Yücetürk
Color Refinement	25%	25%	25%	25%
Branching algorithm	25%	35%	25%	15%
Trees/forests algorithm	20%	–	70%	10%
Preprocessing Twins	–	70%	15%	15%
Fast Part. Refinement	15%	55%	15%	15%
Branching rule/Optimization	–	80%	20%	–
Documentation	70%	10%	10%	10%

Table 8: *Work Distribution Group 28.*

6.2. Team Dynamics

For most of the new algorithms, Jelle and Salih first wrote (or expanded) a pseudocode. After which Minh and Duc worked on actually implementing said algorithm. At the end of the project Jelle set up most of the documentation, which was then filled in.

+ With already discussed pseudocode everyone understand the algorithm and it's easier to implement.
+ Great communication and co-operation between group members.
+ Math student learned some programming hacks from computer science students.
- Because of the differences in coding ability among team members, it was difficult for some to keep up and contribute equally.