2-dimensional Weisfeiler-Lehman implementation for group isomorphism

Karim Elsheikh

February 27, 2020

Introduction:

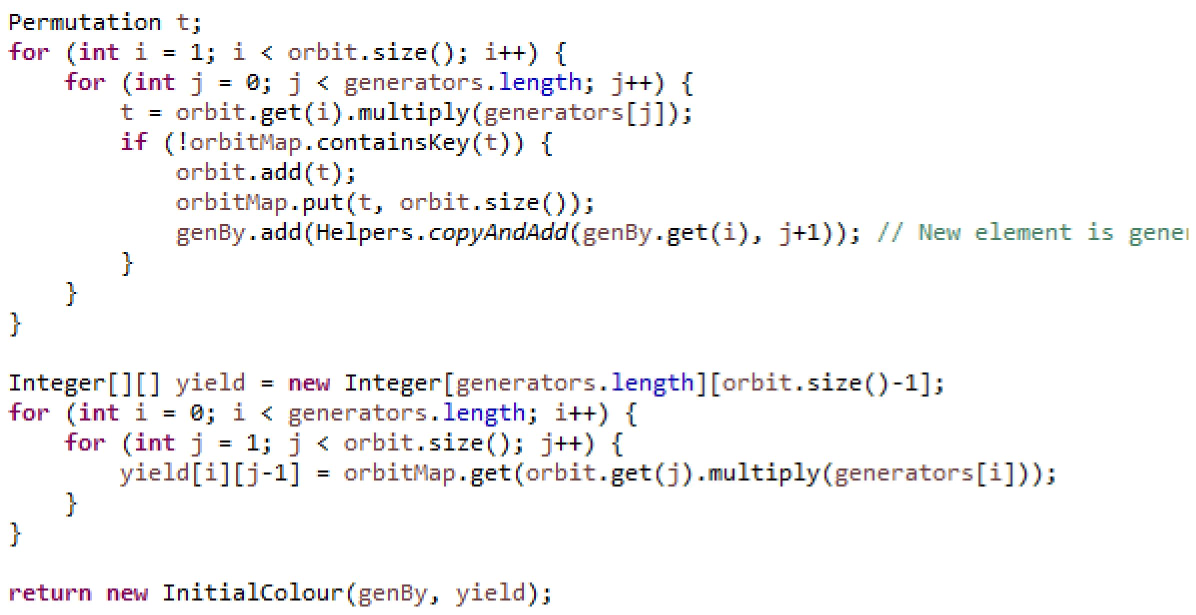
k-dimensional Weisfeiler-Lehman method is a well-known combinatorial method that has been widely studied in the graph isomorphism literature. Essentially, the heuristic used to test graph isomorphism first partitions the set of k-tuples defined over the set of vertices of a graph into color classes and then iteratively for each k-tuple it considers its neighboring k-tuples and changes its color class accordingly. The process eventually stabilizes with a partition of the k-tuples that depends only on the initial color assigned to each of the k-tuples.

The method can also be applied for group isomorphism. After defining a sensible notion to assign an initial color to each k-tuple of a group's elements. The method works as usual, and eventually after running on 2 groups and will decide based on the 2 final partitions of the groups' k-tuples into color classes whether they are non-isomorphic or give a hint that they are similar. The goal was to have an implementation the k-dimensional Weisfeiler-Lehman method for number of dimensions k = 2, and to test it practically on as much groups as possible, while reporting any findings that are of interest.

Implementation:

To implement the method, Java was chosen as a language. We will describe the method that tests isomorphism for 2 input groups given as 2 lists of generators (kWL()), the other method (kWLInv())that generates all the data about a group runs pretty much in the same fashion except it runs on a single group and writes all the information it generates to the storage drive. The GAP Small Groups library was used to provide the groups as input. They were converted to a permutation group using the Sonata package as this implementation only deals with permutation groups.

First the method starts by generating the 2 groups in full using their respective list of generators, then it does a simple check if the group's orders are not equal then we already know they are non-isomorphic. Otherwise, it continues and iterates over every pair of each group and assigns it its initial color. The initial color for each pair was defined as 2 things: 1. a list of minimum sequences of generators that generate each group element, and 2. a list of lists where each list is the result of multiplying a each of the group's elements by a particular generator (corresponding to this list). This initial color captures all the information of the group generated by the pair of generators, in some non-exact sense, it contains already plenty of information about the group.
The following snippet of code is the implementation of the above procedure.



```
Permutation t;
for (int i = 1; i < orbit.size(); i++) {
    for (int j = 0; j < generators.length; j++) {
        t = orbit.get(i).multiply(generators[j]);
        if (!orbitMap.containsKey(t)) {
            orbit.add(t);
            orbitMap.put(t, orbit.size());
            genBy.add(Helpers.copyAndAdd(genBy.get(i), j+1)); // New element is gene
        }
    }
}

Integer[][] yield = new Integer[generators.length][orbit.size()-1];
for (int i = 0; i < generators.length; i++) {
    for (int j = 1; j < orbit.size(); j++) {
        yield[i][j-1] = orbitMap.get(orbit.get(j).multiply(generators[i]));
    }
}

return new InitialColour(genBy, yield);
```

For the rest of the method, the method proceeds by iteratively refining the color classes of each pair by considering its neighboring pairs (i.e., for the pair (i, j) we consider (k, j) and (i, k) for all $k \in$ Group) till it stabilizes. It compares the color classes after each iteration and ends the process abruptly with a non-isomorphic verdict if a difference is found.
Naturally with respect to the data generated by the WL method that depends only on the group given by the list of generators, we wanted to accelerate the process of comparing 2 groups to compare between a large number of groups, so we coded the kWLInv() method to compute the data generated by the WL method and write it to the storage drive. Then the data was collected for all groups of each order in the range 3-255. By defining an ordering over the data and sorting the data, groups having similar data will be in close range. Using that strategy, we found the first examples of non-isomorphic groups who have identical data generated by the 2-dimensional WL method and thus they are not differentiated by the 2-dimensional WL method.

Results:

The first such examples were found in groups of order 128:
The following pairs of IDs are the examples. These IDs are the groups you get by calling
SmallGroup(128, ID) in GAP.
164 and 999
166 and 1014
165 and 1011
174 and 177
173 and 1126
167 and 1013
171 and 1122
831 and 832
555 and 556
807 and 808
For all other orders of 3-255, no pair of undifferentiated groups was found. And to note, for order 256, in the first 8000 groups over 150 pairs were found.

Conclusion:

2-dimensional Weisfeiler-Lehman was implemented, and counter-examples were found in groups of a relatively small order. The number of counter-examples in groups of a certain order appear to depend mostly on the prime powers of the group order. Much more work is needed to detect a pattern (if any) in the groups that 2-WL can't differentiate. 3-WL and WL for higher dimensions is an obvious continuation to this project, and it will be interesting to find in which orders the counter-examples for these higher dimensions lie (Or if some dimension is optimal).