The Supercharacter Theory Finder

Shawn Burkett, Jonathan Lamar, Megan Ly, Nat Thiem

Contents

1 What is it?

The Supercharacter theory finder is a sage script which computes all supercharacter theories of a group from its character table. As the script is written, the group must be constructible in magma (there is a version of `sct_finder` for GAP: the magma version is more robust but can only be used on euclid, while the GAP version has more terse output, but runs on any computer). After the computation is complete, the output in its most raw form will be a list of partitions of $\mathrm{Irr}(G)$ (really of `range(len(rows(CT)))`) and a list of partitions of $\mathrm{Cl}(G)$ (really of `range(len(columns(CT)))`) in the same order, so that we can index them simultaneously: e.g., if `X` is the former list and `K` the latter, then `(K[i],X[i])` is the `i`-th supercharacter theory. The output also includes a sage `LatticePoset` object whose elements are the indices of the supercharacter theories and whose order is the refinement order. Finally, the output also includes some canonical subposets of the full lattice.

2 How does it work?

2.1 The algorithm

The supercharacter finder follows an algorithm based on [Hen08, Appendix A]. The strategy is to build partitions of $\mathrm{Irr}(G)$ part-by-part, checking at each step whether the parts are "supported" by their generated subalgebra. We state the algorithm more precisely with the following terminology. Define the *Wedderburn sum* of a subset $X \subseteq \mathrm{Irr}(G)$ to be the character $\sigma_X = \sum_{\chi \in X} \chi(1)\chi$ and define a *partial partition* to be a collection of pairwise disjoint nonempty subsets of $\mathrm{Irr}(G)$. Finally, define the *support*,[1] $\mathcal{S}(\mathcal{X})$, of a partial partition $\mathcal{X}$ to be the partition of $\mathrm{Irr}(G)$ corresponding to the equivalence relation $\chi \sim \psi$ if the coefficients of $\chi(1)\chi$ and $\psi(1)\psi$ agree in all elements of the subalgebra generated by the Wedderburn sums of the parts of $\mathcal{X}$. By [BLLW17, Lemma 5] (which just restates [Hen08, Lemma A.4]), it follows that a partial partition which is *not* a refinement of its support[2] will not yield a supercharacter theory. Moreover, a full partition $\mathcal{X}$ of $\mathrm{Irr}(G)$ yields a supercharacter theory if and only if $\mathcal{X} = \mathcal{S}(\mathcal{X})$.

Our strategy is recursive. If $\mathcal{X}$ is a partial partition which is admissible in the sense that each part of $\mathcal{X}$ is a subset of a part of $\mathcal{S}(\mathcal{X})$, then we pick some character ψ in a nonrandom fashion not contained in any part of $\mathcal{X}$, and check for admissiblity all partial partitions $\mathcal{Y}$ of the form $\mathcal{X} \cup \{Y\}$, where Y contains ψ and is disjoint from the parts of $\mathcal{X}$. This is made precise as follows.

Algorithm 2.1.

1. Let $\mathcal{X}$ be a partial partition with support $\mathcal{S}(\mathcal{X})$ and suppose each part of $\mathcal{X}$ is a subset of a part of $\mathcal{S}(\mathcal{X})$.
2. Let ψ be a character not contained in any part of $\mathcal{X}$.
3. Let S be the part of $\mathcal{S}(\mathcal{X})$ containing ψ.
4. For each subset Y of S containing ψ, do:
   (a) Let $\mathcal{Y} = \mathcal{X} \cup \{Y\}$.
   (b) If each part of $\mathcal{Y}$ is contained in a part of $\mathcal{S}(\mathcal{Y})$, then:
      i. If $\mathcal{Y}$ is a full partition, record it as a supercharacter theory.
      ii. Else, call this algorithm recursively with $\mathcal{Y}$ in place of $\mathcal{X}$.
   (c) Else, return.

It is not hard to see that Algorithm 2.1 returns all supercharacter theories containing the parts of $\mathcal{X}$ as supercharacters, and consequently, calling this algortihm on $\{\{\mathbf{1}_G\}\}$ will return all supercharacter theories. The choice of a character ψ to lie in the new

[1] Hendrickson called this the *filtration* on the class sum side. Somehow the terminology changed along the way.

[2] i.e., if each part of the partial partition is a subset of a part of the support.

part Y eliminates repetition, so supercharacter theories are only found once (this also cuts down on run time significantly). Moreover, we also cut down on run time by selecting only subsets of the part of $\mathcal{S}(\mathcal{X})$ which contains ψ. This will be a necessary condition for $\mathcal{Y}$ to be admissible, since one can check that

$$\mathcal{S}(\mathcal{Y}) = \mathcal{S}(\mathcal{X}) \wedge \left(\bigwedge_{X \in \mathcal{X}} \mathcal{S}(\{X, Y\}) \right). \tag{2.1}$$

2.2 The implementation

Unfortunately, this algorithm is both recursive and requires some memory of past iterations. We keep track of the parts of $\mathcal{X}$ using a list of sage `Set` objects called `subsets`. We also keep a list of sage `SetPartition` objects called `previous_supp`. At each stage of the program, the final entry of `previous_supp` is the support of $\mathcal{X}$. These variables are both used like stacks in practice, so we append supercharacter theory partitions to the variable `sct_list` and when returning back (up one level), we pop `subsets` and `previous_supp`. The source should pretty readable with this in mind and given Algorithm 2.1.

3 How do I use it in my research?

Load `sct_finder.sage` (which loads the magma `sct_finder` class) and create an instance of the class with something like `G = sct_finder('magma_command','myGroup')`, where `'magma_command'` is the string which constructs your group in magma (see below for more complicated groups), and `'myGroup'` is a string which you can use as a label for the group if you are storing the data for later. For instance, your magma command might be `'AlternatingGroup(5)'` or `'DirectProduct(CyclicGroup(2),CyclicGroup(4))'` and the nickname could be `'A5'` or `'Z2xZ4'`. In order to compute the supercharacter theories of this group, execute `G.go()`. Note this only works if you have magma on the machine you're using (you could use the GAP version, but just pony up and ssh in to euclid). Once the script completes (assuming it does), the output will all be accessible as public methods and variables of `G`. There is an important method, `G.export_data()`, which saves the output in several files in whatever directory you are in. Since these take so long to compute, you should always export data if you plan to look at a particular group a lot.

Suppose you are computing your group in several steps, e.g., as a semidirect product or by loading some magma group library. You can do these first in sage using `magma.eval`. For example, to construct a semidirect product, you might do the following commands.

```
>load('sct_finder.sage')
>magma.eval('A<x>:=CyclicGroup(19)')
>magma.eval('B<y>:=CyclicGroup(3)')
```

```
>magma.eval('phi:=Hom<B->AutomorphismGroup(A)|y:->Hom<A->A|x:->x^7>>')
>magma.eval('G:=SemidirectProduct(A,B,phi)')
>myGroup=sct_finder('G','myGroup')
```

Suppose you have a bunch of groups, say $\{G_1, G_2, ..., G_n\}$ that you would like to analyze. As mentioned above, you could open sage, load `sct_finder.sage`, and manually compute each one. You should write a quick script to run examples for you on a server, save them all in one folder with a uniform naming convention, and write another script to load everything in that folder for analysis. If there are methods from the `sct_finder` class you'd like to use later, then copy them from the source to your script and mutatis mutandis them until they work. I have included an example dataset in the folder `example_dataset` for you to see how all of this can be done, but obviously you should modify it as you see fit. Any of the scripts in that folder can be used as a jumping-off point, but in particular look for the files `analyze_posets.sage` and `examples/large_example.sage`.

4 How can I improve it?

First and foremost, you should fix the version of `sct_finder` in the 'unfinished version' folder. There is a bug in the current version where the group and its character table are computed once in magma at the beginning of the script to compute the supercharacter theories, and then again near the end to compute the subposets of automorphic, galois, and characteristic subposets. For most groups, this is no problem. However, some groups, e.g., the unipotent upper triangular groups `SylowSubgroup(GL(n,q),p)` are computed somewhat stochastically by magma (I don't fully understand how) and therefore often get returned only up to conjugacy in some larger group. The extremely unfortunate result is that these groups' character tables have somewhat randomly permuted rows and columns. Thus, the program can return incorrect data for the canonical subposets (but the supercharacter theory lattice is always correct). The unfinished version is designed to fix this problem by saving the character table and other critical info locally and reloading it when needed, but it is a nightmare of syntax errors.

Of secondmost importance, you should implement the main algorithm in a better language. I (Jon) tried doing this in C++ (the source code is included, but it's full of memory leaks) for groups with integral character tables. You should try it in Haskell. This is a good language for recursive algorithms.

Finally, go forth and explore. The lines between math and programming are blurred here, since improving the program is nearly equivalent to doing more supercharacter theory research. Find new subposets! Prove that the subposets of automorphic and Galois supercharacter theories are meet-closed so you can uncomment those two lines!

References

[BLLW17] Shawn Burkett, Jonathan Lamar, Mark L. Lewis, and Casey Wynn. Groups with exactly two supercharacter theories. *Communications in Algebra*, 45(3):977–982, 2017.

[Hen08] Anders Hendrickson. *Supercharacter theories of finite cyclic groups*. PhD thesis, University of Wisconsin – Madison, 2008.