Tyler Cowan's Guide to Modifying the ECJ statistic files.

This is a quick guide to show you how to modify the default ECJ statistic files. The final goal is to leave the console output as is while customizing the job.x.out.stat files to facilitate your specific requirements.

Before starting, make sure you've followed Illya Bakurov's Symbolic Regression Tutorial.

I will be using ECJ23 and NetBeans IDE8.2, but the general idea should be applicable to any versions.

Assuming you have a functional Symbolic Regression project, your statistic files will probably look like:

```
Generation: 0
Best Individual:
Subpopulation 0:
Evaluated: true
Fitness: Standardized=4.220966144E9 Adjusted=2.369125848556172E-10 Hits=0
Tree 0:
 (* (* 1.0 x) (* x x))

Generation: 1
Best Individual:
Subpopulation 0:
Evaluated: true
Fitness: Standardized=4.137563136E9 Adjusted=2.4168815481207725E-10 Hits=0
Tree 0:
 (* (* (* (* 1.0 1.0) (- x 1.0)) (- (/ 1.0
     1.0) (* 1.0 x))) (- (sin (* x x)) (+ (* 1.0
     x) (cos x))))
```

But this not helpful if you're looking for detailed data to use in something like Excel.
In this case, I prefer my output to be tab-delimited and contain only relevant information, but you might have something else in mind. In either case, the solution involves overriding the SimpleStatistics file with your own. Here are the steps:

Step 1. Open your ecj.x.jar or download the source code that corresponds to the ecj.x.jar file you are using. At the time of writing this, the source of each version can be obtained at https://cs.gmu.edu/~eclab/projects/ecj/.

Step 2. Open the source code and navigate through ecj > ec > simple and you will see SimpleStatistics.java. This contains the instructions ECJ uses to print stat files. Find the postEvaluationStatistics() method and copy it.

Step 3. Go to your Symbolic Regression project and make new class in main. I called mine CustomStatistics.java.

Step 4. Extend the class to SimpleStatistics and add the import for it.

Step 5. Paste the postEvaluationStatistics() into your CustomStatistics class as an override.

Step 6. Add any remaining required imports and add "boolean warned = false;" at the top of the postEvaluationStatistics() method.

Your CustomStatistics class should now look like this:

```
package main;

import ec.EvolutionState;
import ec.Individual;
import ec.gp.GPIndividual;
import ec.gp.GPNode;
import ec.gp.koza.KozaFitness;
import ec.simple.SimpleStatistics;

public class CustomStatistics extends SimpleStatistics {

    boolean warned = false;

    @Override
    public void postEvaluationStatistics(final EvolutionState state) {
        .
        .
        .
```

Step 7. Near the bottom of the postEvaluationStatistics() method, you'll see the following:

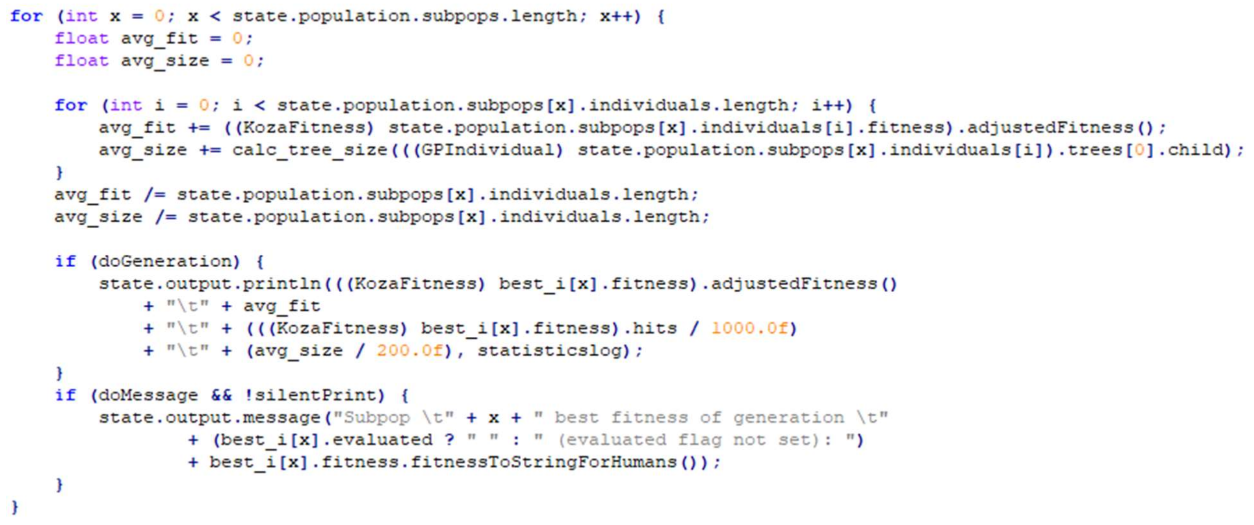
```
for(int x=0;x<state.population.subpops.length;x++)
    {
    if (doGeneration) state.output.println("Subpopulation " + x + ":",statisticslog);
    if (doGeneration) best_i[x].printIndividualForHumans(state,statisticslog);
    if (doMessage && !silentPrint) state.output.message("Subpop " + x + " best fitness of generation" +
        (best_i[x].evaluated ? " " : " (evaluated flag not set): ") +
        best_i[x].fitness.fitnessToStringForHumans());

    // describe the winner if there is a description
    if (doGeneration && doPerGenerationDescription)
        {
        if (state.evaluator.p_problem instanceof SimpleProblemForm)
            ((SimpleProblemForm)(state.evaluator.p_problem.clone())).describe(state, best_i[x], x, 0, statisticslog);
        }
    }
}
```

I modified it to look like this:

```
for (int x = 0; x < state.population.subpops.length; x++) {
    if (doGeneration) {
        state.output.println("CUSTOM OUTPUT GOES HERE", statisticslog);
    }
    if (doMessage && !silentPrint) {
        state.output.message("Subpop \t" + x + " best fitness of generation \t"
                + (best_i[x].evaluated ? " " : " (evaluated flag not set): ")
                + best_i[x].fitness.fitnessToStringForHumans());
    }
}
```

And within the "CUSTOM OUTPUT GOES HERE" can be whatever you need. For example, I wanted a tab-delimited output to show the best adjusted fitness, average fitness, best hits, and average tree size of each generation. That looked like:



```
for (int x = 0; x < state.population.subpops.length; x++) {
    float avg_fit = 0;
    float avg_size = 0;

    for (int i = 0; i < state.population.subpops[x].individuals.length; i++) {
        avg_fit += ((KozaFitness) state.population.subpops[x].individuals[i].fitness).adjustedFitness();
        avg_size += calc_tree_size(((GPIndividual) state.population.subpops[x].individuals[i]).trees[0].child);
    }
    avg_fit /= state.population.subpops[x].individuals.length;
    avg_size /= state.population.subpops[x].individuals.length;

    if (doGeneration) {
        state.output.println(((KozaFitness) best_i[x].fitness).adjustedFitness()
            + "\t" + avg_fit
            + "\t" + (((KozaFitness) best_i[x].fitness).hits / 1000.0f)
            + "\t" + (avg_size / 200.0f), statisticslog);
    }
    if (doMessage && !silentPrint) {
        state.output.message("Subpop \t" + x + " best fitness of generation \t"
                + (best_i[x].evaluated ? " " : " (evaluated flag not set): ")
                + best_i[x].fitness.fitnessToStringForHumans());
    }
}
```

Step 8. Go to your simple.params file and change

stat = ec.simple.SimpleStatistics

to

stat = main.CustomStatistics

Step 9. Done!

There may be some discrepancies. Use this as a general guide.

I’ve attached an example of CustomStatistics below.

```
package main;

import ec.EvolutionState;
import ec.Individual;
import ec.gp.GPIndividual;
import ec.gp.GPNode;
import ec.gp.koza.KozaFitness;
import ec.simple.SimpleStatistics;

public class CustomStatistics extends SimpleStatistics {

    boolean warned = false;

    @Override
    public void postEvaluationStatistics(final EvolutionState state) {

        // for now we just print the best fitness per subpopulation.
        Individual[] best_i = new Individual[state.population.subpops.length];  // quiets compiler complaints
        for (int x = 0; x < state.population.subpops.length; x++) {
            best_i[x] = state.population.subpops[x].individuals[0];
            for (int y = 1; y < state.population.subpops[x].individuals.length; y++) {
                if (state.population.subpops[x].individuals[y] == null) {
                    if (!warned) {
                        state.output.warnOnce("Null individuals found in subpopulation");
                        warned = true;  // we do this rather than relying on warnOnce because it is much faster in a tight loop
                    }
                } else if (best_i[x] == null || state.population.subpops[x].individuals[y].fitness.betterThan(best_i[x].fitness)) {
                    best_i[x] = state.population.subpops[x].individuals[y];
                }
                if (best_i[x] == null) {
                    if (!warned) {
                        state.output.warnOnce("Null individuals found in subpopulation");
                        warned = true;  // we do this rather than relying on warnOnce because it is much faster in a tight loop
                    }
                }
            }
            // now test to see if it's the new best_of_run
            if (best_of_run[x] == null || best_i[x].fitness.betterThan(best_of_run[x].fitness)) {
                best_of_run[x] = (Individual) (best_i[x].clone());
            }
        }
                                            // main loop, prints once per generation
        for (int x = 0; x < state.population.subpops.length; x++) {
                                                            // initializing average values
            float avg_fit = 0;
            float avg_size = 0;
            // calculated average values for this generation
                                                            for (int i = 0; i < state.population.subpops[x].individuals.length; i++) {
                avg_fit += ((KozaFitness) state.population.subpops[x].individuals[i].fitness).adjustedFitness();
                avg_size += calc_tree_size(((GPIndividual) state.population.subpops[x].individuals[i]).trees[0].child);
            }
            avg_fit /= state.population.subpops[x].individuals.length;
            avg_size /= state.population.subpops[x].individuals.length;
                                                            // printing information to job.x.out.stat
            if (doGeneration) {
                                                                        state.output.println(((KozaFitness) best_i[x].fitness).adjustedFitness()
                                                                                    + "\t" + avg_fit
                                                                                    + "\t" + (((KozaFitness) best_i[x].fitness).hits / 1000.0f)
                                                                                    + "\t" + (avg_size / 200.0f), statisticslog);
            }
            if (doMessage && !silentPrint) {
                state.output.message("Subpop \t" + x + " best fitness of generation \t"
                        + (best_i[x].evaluated ? " " : " (evaluated flag not set): ")
                        + best_i[x].fitness.fitnessToStringForHumans());
            }
        }
    }
   // additional method added to calculate tree size. should be called with the tree's root as node.
    public int calc_tree_size(GPNode node) {
        int c = 1;
        for (GPNode n : node.children) {
            c += calc_tree_size(n);
        }
        return c;
    }
```