Reminder:

1. Why Gnuplot is needed?
   1. We know Fink is the OSx package manager that allows us to acquire the needed resources for Gnuplot.
2. Legalization problem
   1. Able to adjust the AR/Size if any block doesn’t fit.
   2. The current container is able to inform its parent to adjust the size as well to compensate the change if needed.
3. Detailed plan to understand the algorithm
   1. Go through the algorithm line by line
      1. This will eventually allows us to understand what are the uses for each class and its functions for the FP algorithm.
      2. Write notes/comments while doing so.
      3. Discover the possible adjustments that can be made to solve the legalization problem.

Some Info:

1. geogLayout represents Cluster Type.
2. gridLayout represents Grid Type.
3. bagLayout: Group Type.
4. fixedLayout: scaled Group Type.

Next Step: Solving Legalization Problem

1. Look closely (step-by-step) at a successful case
2. Look closely (make up) at a failing case
3. The key lies in layout methods
   1. Grid
   2. Bag
   3. Fixed
   4. Geog
   5. FPCompWrap
4. Also look at the Layout Helper
   1. How are center items defined?

Use GDB to look at the values of these variables.

Add our own layout helper. (not touch the original one)

In my opinion, the fastest way to learn someone else´s code, (especially when changes triggers unexpected behavior as you describe) is to **step through the code using a debugger**.

Begin with stepping through what seems to be the main loop / main methods of the program. Use the **step into** and **step out** functions to see what different methods does. This will teach you the general structure of the code.

After that, divide and conquer by stepping through and learning the different parts of the program on a deeper level. In most debuggers you can **study variables** and their **current values**. Study how they change and when.

Set out **breakpoints** on methods that trigger behaviors that concerns you. For example if you are trying to change a text in the program, and the text keep changing back to the original value, set breakpoints on all places where the text is changed, or try to move all these changes to one single method. Use the **call stack** to see from where this method is called, etc etc.

If changing a line of code causes **unexpected changes** on other places, put a breakpoint on that line, and see what happens there, for example by checking the values of current variables in scope, using step into, or the call stack to see from where the call came.

By doing this alot, you will start to learn the structure of the code surprisingly fast. I started out just like you did on my first programming jobs, with lots of code that had been written many years ago and been changed by many people through many years. The code was not mine only since there where other people working on it at the same time. I couldn't rewrite it all at that point. Writing tests for all that code would have taken me months or years. The debugger really saved me, don't know how I would have learned the code without it...