1 Getting Started

1. Go ahead and download the latest stable version of Netbeans from the Netbeans website. You only need the Java SE version.

   Regardless of what you've used in the past, you really need to use Netbeans for this, because otherwise, you'll have to edit the auto-generated gui code which will become an unmaintainable nightmare. We've noticed subtle incompatibilities between the GUI builders in different versions of Netbeans, so make sure that everyone is using the same version. Also, you may want to use your own computer for this, as the Jane project takes up a lot of space and you'll hit your quota very quickly if you do development on a CS department machine, unless you get more quota.

2. If you haven't installed the JDK yet, do so now. Make sure you are using the 64 bit version as Jane is about 2.5 times faster when running inside a 64 bit JVM.

   Also, I recommend that you install both JDK 5 and JDK 6. (I think JDK 7 is slated to come out this Summer, so you might use that instead of 6 if you want to.) You need at least JDK 6 to run Netbeans. However, you should set the JDK version used for compiling and running the project in the project preferences to JDK 5.

   The reason for this is as follows. Right now, we support running Jane on JDK 5. There isn't really a good technical reason to restrict Jane to a higher version, and lots of Jane's users run antiquated technology, so it seems reasonable to support Java 5. Note that if you do decide to drop support for it though, you can get rid of the TableSorter class, because Java 6 includes similar functionality and you can remove the uses of reflection in the Utils class.

   You don't have to install JDK 5 because JDK 6 can target Java 5 (and the project is currently set to do this), but the problem is that even if you target Java 5, you still use Java 6's class libraries during compilation. So, if you wind up using a method or class that is in JDK 6 but not 5, the compiler won't complain, but your application will break if you try to run it on Java 5 since that class or method won't be present. Using JDK 5 for compiling and running Jane during development prevents this from occurring, which is why I recommend it.

3. Install Netbeans. Then start it.

4. You may see a globe with a yellow arrow on it in the bottom right hand corner of the Netbeans window shortly after starting. This means that there are updates available. You can click on that or go to tools > plugins to install the updates.

5. After installing updates, reboot and check in the aforementioned dialog to see if there are any more updates and keep rebooting and installing until there are no more updates.

6. Now we need to setup subversion. If you aren't on Windows, things will probably work out of the box. If you are using Windows, you'll need to download and install an SVN client. (Netbeans comes with a bundled subversion client for Windows, but it isn't compatible with the 64 bit JVM.) You can get that here.

7. Go to team > subversion > checkout. Input the path to the subversion bin directory when prompted. Then select the same menu item again and you should be brought to a wizard. The repository url is https://svn.cs.hmc.edu/svn/jane. Use your username and password that you were given. Specify a folder to check out everything into, then hit finish. (Don't specify a folder to check out, because we want everything.)

8. Netbeans should automatically open the Jane project. At this point, you should be able to hit the green arrow at the top of the Netbeans screen to run Jane. Note that you should make sure that the project is setup to compile on save as this makes fixing syntax errors a lot easier. Just google compile on save to figure it out.

9. You may also wish to install TortoiseSVN (link here. It has nice shell integration and will help you in managing files outside of the Netbeans project folder.

2 Useful Netbeans Features and Tips

Netbeans has several useful features. Here are a few you may wish to familiarize yourselves with.

1. If you control/command click on a variable or method, you'll be taken to where it was declared. There are also back and forward buttons at the top of the editor window.

2. Netbeans has built in debugger integration and profiling. They are respectively the buttons next to the green triangle.

3. Netbeans has some refactoring support. Right click on a file in Netbeans and go to Refactor. A commonly used one is renaming a class and automatically having all java files that use it updated to use the new name.

4. Netbeans includes a feature called local history. It keeps track of changes you make to files over the past 7 days by default. You can diff against and revert to previous versions of files, even if you never checked them in to SVN. You can access local history by right clicking a file and selecting local history.

5. Make sure you never ever have two people modify any of the GUI files at the same time. Otherwise, you'll both end up causing changes to the corresponding .form files which will not merge properly and you'll have to

lose one set of changes. This applies to anything that when you open it, there is both code view and a design view

6. In general, the GUI builder is very finicky, so make sure you run the application and try resizing it every time you make a set of changes in Design view to make sure that nothing got messed up.

7. In order to build the redistributable packages for Jane, go to the files tab in Netbeans, then right click build-packages.xml. Go to run target, then click clean-package-all. Your redistributable files will be in the packages directory. The file Website.zip contains the website with the redistributables in the downloads folder.

8. If you right click on a class name, method name, or variable and hit find usages, it will show you all the places in your project where it is used.

3 Folder Structure

At the top level, there are four folders: data, doc, Jane, and original-images. Data contains various real trees for use in testing Jane. Doc contains documentation regarding the DP as well as this file. Jane is the Netbeans project with all the source code. Also, note that the Jane website is stored in the Website subdirectory of this folder. Finally, original-images contains some of the artwork used in Jane. Note that the jane.* files in this directory were created in Adobe Illustrator. You should use Adobe Illustrator to edit the AI file, then export as an SVG. Otherwise, you run into issues with the shadows not being quite right.

Now, we move on to subdirectories of the Netbeans project folder, Jane. I've already mentioned the website folder.

1. Build and Dist are created by Netbeans for use in compiling.

2. Images contains the icons that Jane uses. These were generated from the aforementioned jane.ai file.

3. Lib contains a JAR for the fastlib library some of whom's classes are used in the DP.

4. Nbproject contains various project configuration files. You probably won't need to mess with this.

5. Packages is a directory which is used by my packaging script for where to put the redistributable packages.

6. Scripts contains various scripts for using Jane including a handy one for distributing the computation across a large number of computers. This is what was used for running Jane on a 168 node cluster in 2010.

7. Src contains all of the source code.

8. Test contains the source code for any unit tests you may wish to create. In particular, I would encourage you to make a set of test timings and trees for the DP (which is arguably the most important part of the program) to make sure that any changes made don't break anything. There is an example test file there already that you are welcome to use.

9. There are also a few files in this folder. ABOUT.txt is a simple about file that gets included in all of the packaged files. LICENSE.txt is also included in these files.

10. Build.xml probably shouldn't be touched, except to change the application.version property before you do releases.

11. Manifest.mf and build-before-profiler.xml can also be left alone.

12. Build-packages.xml is an Apache Ant build script that handles compiling everything, optimizing and shrinking it using ProGuard, packaging it into an app bundle for Mac, an exe for windows, etc, and zipping everything up, then zipping up the contents of the Website directory with the bundle/exe/etc in the downloads folder. You can then give packages/website.zip to Prof. Ran, and he can update the website.

4 General Tips and Things You Should Know

1. You may notice that Jane is unnaturally fast on certain very small trees (eg. the Gopher-Louse tree). Jane caches solutions to the 4000 most recently solved timings and uses those instead of resolving if possible. As a result, you should not benchmark Jane against trees this small.

2. Java's GUI framework is called Swing. Swing is actually a single threaded framework, which means you don't have to worry about weird concurrent access stuff when writing your gui code.

   However, all Swing code needs to run in something called the Swing Event Dispatch Thread. If it doesn't, strange, difficult to reproduce errors can pop up. So, if you want to update something in the gui by calling some method but your current code is not in the event dispatch thread, you can run the code you want in the dispatch thread using SwingUtilities.invokeLater or SwingUtilities.invokeAndWait

   So, for example, code that is in estimate_time_buttonActionPerformed method of Design.java will be run in the Event Dispatch thread because that method is only called by the event dispatch thread. As such, you do not need to use SwingUtilities there. However, the thread that gets started from that method obviously is not the Swing Event Dispatch thread, so when that thread updates the GUI, it does need to use SwingUtilities.

   Also, since Swing is single threaded, if you call some long running method from your GUI code, it will freeze the entire GUI until it completes. Don't

do this. Instead, create a new thread and run the code inside that thread. For an example, see the runSolve() method in Design.java.

Note that we haven't always done the best job of following these practices. You will see code that should be run as a separate thread, and you will see code that should be invoked using SwingUtilies, but isn't. I encourage you to rectify such problems when they become apparent.

3. You may (or more likely, probably don't) wonder why the dp table (in the 3D case) is stored as a 3D array such that table[time][e_P][e_H] is used rather than table[e_P][e_H][time] or some other permutation. The reason is CPU cache. Unlike C which uses pointer manipulation to achieve the illusion of multidimensional arrays, Java's 3D arrays are actually arrays of arrays of arrays. Since the outer loop of the DP is time, the loop inside of that is over parasite edges, and the loop inside of that is over host edges, the ordering we chose means the CPU operations will always be restricted to a subarray of the DP table at any given time which makes more efficient use of cache, since you don't have to swap in and out different subarrays as much. If this doesn't make sense, ask Prof. Ran for more info.

4. You may notice that two solutions in the solution table appear the same in solution viewer. The solution table stores timings which will be different, but the solution viewer displays the best embedding corresponding to that timing. Two different timings may have the same best embedding, which causes the observed similarity.

5. Java doesn't have a built in infinite value for ints, so we use 999999999 to represent an infinite cost and 88888888 to represent an infinite distance for host switching purposes. If you see either of these numbers, that is where they came from. Also, note that we perform checks whenever we do arithmetic such that if you for example add 1 to infinity, you get infinity (999999999) back and the same thing happens if you subtract 1.

6. If you haven't heard the phrase postorder before, it means you traverse a tree in such a way that you visit a node's children before you visit the node.

7. If you look around the DP, you'll see a lot of references to something called a costinfo which is a long. Depending on which DP table is being used, this may hold the cost, event counts, and other information necessary to reconstruct a solution.

8. If you aren't familiar with Java's data types, ints are 4 bytes and longs are 8 bytes. This is all regardless of what architecture you are running on.

9. Jane uses a thread pool to do solves. By default, it allocates twice as many threads as the system has cores (to take advantage of the ability for one thread to run while another is context switching). It uses the Java 5 concurrency API, specifically, the Executor interface.

10. Any threads used in thread pools need to be Daemons. Otherwise, they prevent the CLI from exiting properly.

11. Don't rewrite the solution viewer. Refactoring is fine, but one of us rewrote it in 2010 and it was all he did for weeks. There are lots of special cases, and it is really a mammoth of a project.

5 Java Packages

Jane is divided up into several packages. This section provides an overview of them.

1. components

   This package contains the TableSorter class from oracle.

2. edu.hmc.jane

   The command line interface (CLI) is in here along with most data structures used in Jane (with the exception of the DP tables). The Tree class contains among other things a method for generating random timings.

3. edu.hmc.jane.gui

   This contains the various forms. Of particular note is the JSliderInput class. You can drag this into the GUI builder. It provides a nice slider and text box combo that validates its input. Design is the class that corresponds to the main window

4. edu.hmc.gui.light

   The solution viewer has to display $O(n^2)$ components. When we used Swing components to do this, it was horribly slow. John ultimately created a partial reimplementation of Swing that (since it doesn't have as many features) runs substantially faster for our purposes. This is contained in this package. Note that there are a few differences between Swing and the various LightComponents in this package. For example, when you click on a light component, light components under that one also receive the event, unlike Swing.

5. edu.hmc.jane.io

   These classes deal with reading tree files and mappings as well as timings.

6. edu.hmc.jane.solving

   Everything dealing with the DP is in here. ArrayDP3 is the logic of the DP while DPTable and its subclasses provide the data structure that stores the table itself. ArrayDP3 can determine just the cost of the optimal embedding, the cost and event counts, or all information about the optimal embedding necessary for it to be reconstructed.

The various classes ending in HSS are host switch selectors. Basically, they tell you where the best place to host switch to is. The reason we have these is because the host switch selection process is algorithmically different depending on whether you limit host switch distance and use regions or not.

The various Solver classes are fairly thin wrappers around ArrayDP3 they set up ArrayDP3 with whatever dp table it needs and let it do its thing. They also provide a mechanism for retrieving the results of the solve. You can pass these directly to an Executor since they implement the Callable interface.

Generation and Heuristic contain methods for breeding, evolving, and evaluating populations of timings.

7. edu.hmc.jane.util

   This contains various utility classes. The RandomBufferedArrayList is pretty cool. There was actually at one point in time when a Google interview question was based on the principle that it uses.

8. org.apache.commons.math.special

   This contains the Gamma class which is used in stats mode for generating random trees according to the Yule model.

6 Feature, Refactor, and Bug Fix Ideas

Here are a few simple features and bug fixes you may wish to consider adding, as these will help you learn the code base.

1. A file read by Jane may mistakenly specify in its mapping an association that includes an internal node. All parasite-host associations should be between tip nodes. At the moment, Jane will try to store this association as it does any other. This may cause Jane to crash or, even worse, think nothing is amiss until the user tries to solve the instance at which point Jane will likely crash. In the latter case, the cause of the crash will be difficult to identify. When reading in files you should make sure that the mapping given contains only tip nodes.

2. The host regions feature was never thoroughly tested. There is a chance there are issues with the functionality in the version of Jane currently in SVN. You should probably test this feature thoroughly and fix any bugs that arise.

3. When no solutions are found during solve mode (perhaps time zones coupled with limited host switch distances prevent a valid mapping between the trees), Jane informs the user and returns to normal. But Jane crashes whenever no solutions are found during statistics mode. This should be fixed.

4. The BufferedHashMap class can probably be rewritten to extend LinkedHashMap and overwrite the removeEldestEntry(Map.Entry) method. It currently duplicates functionality already present in Java.

5. Right now, the setRunStatus method in Design.java is a bit of a mess. What it really should do is have a list of elements to disable, then loop over them, rather than just calling setEnabled on them individually as it does now.

6. It would be nice if there were a way to cancel a run without exiting the program. For example, you might change the go button to a cancel button and prompt the user if she is sure she wants to cancel. You would need to add logic to threads in the thread pool to gracefully terminate.

7. The time estimation feature is currently only present in the GUI. It would be nice if it were present in the CLI.

8. The time estimation feature tends to overestimate the amount of time needed, especially if you have a lot of cores at your disposal. The problem is that the JVM only compiles methods that are invoked frequently and take a lot of time, so it is difficult to quickly get an estimate of the time needed quickly.

9. The progress bar for stats mode is pretty coarse. It only tells you how many samples have finished. It would be nice if it were based on the number of solves like the progress bar used when doing simple genetic runs. You would need to modify hte statsProgressThread method in Design.java. Better yet would be to combine the statsProgressThread and solveProgressThread methods.

10. Jane includes the ability to save images of the contents of the solution viewer as well as the histogram. However, right now, it simply saves it at the size it is being displayed on the screen. It would be nice if users could select the size.

11. There are some instances of code duplication that could be eliminated by modify the build-packages.xml script. Specifically, the options for the JRE are duplicated in all of the scripts in the scripts folder even though they are specified in build-packages.xml.

12. Right now, it can be hard for a user to tell which version of Jane she is running. An about dialog with a version number would fix this. You could also add the version number to the title bar. However, to avoid code duplication you shouldn't manually enter the version number. Instead, you should load it from a text file in the jar as a jar resource (google jar resource and look in the Utils class). You can have the build script automatically create the text file at build time and put it in the JAR.

13. Jane doesn't currently remember what the last folder you opened a file from was after you close and reopen the program. Swing includes built in utilities for handling this sort of persistence, but I'm not sure what they are called or how to use them.

14. We partially refactored the Heuristic class into Generation, but the refactor is not complete. You could finish this by moving the majority of the remaining functionality of Heuristic into Generation.

15. At present, Jane won't find solutions if a parasite has a time zone before the time zone of the host root. What it should do is force that parasite onto the edge before the host root.

16. The whole LightComponent thing is a pretty hackish solution. It would be great if there were a way to use only pure Swing components and get rid of the LightComponent classes without negatively impacting the performance of the solution viewer.

17. Input validation is currently duplicated in the GUI and CLI. You could refactor it such that it is all handled from one place.

18. The .nex files generated by CoRe-PA contain a lot of information that Jane cannot currently read. At the very least, it would be nice if Jane could read in the time zone information present in these files. Jane does try to read time zone information but it does so in the same way it reads time zone information from other .nex files. CoRe-PA .nex files, however, list time zones in a method similar to .tree files, and thus Jane is unsuccessful at reading this data.

19. Jane will usually crash if the host or parasite tree has only one node. While these cases are trivial as far as actually solving them is concerned, they show up when people wish to test Jane on randomly generated trees. For completeness, it would be nice if Jane could handle these cases.

20. See the suggestion for DP tests in the Folders section.