An introduction to SAGElyzer

Jianhua Zhang

April 25, 2007

1 Introduction

SAGElyzer is a system for SAGE data management, analysis, and annotation built upon interfaces built using R tcltk. The functionalities are never complete but can be expanded easily when needed. At the time of this release (1.3.0), *SAGElyzer* allows users to manage data from SAGE libraries, analyze data, and annotate SAGE tags identified by the analysis.

As SAGE libraries are potentially large, a database is required to support data storage and retrieval. The current version of *SAGElyse* has been tested against a PostgreSQL database under both Windows and Unix. However, *SAGElyzer* is supposed to work together with any database management system as long as a connection to the database can be made.

2 Setting up a database

When database management system has been chosen, read the manual or tutorials in the Intenet for setting up the database. For a PostgreSQL database on linux, readers are referred to `http://techrepublic.com.com/5100-6261-1054332.html` for the procedures involved to set up a database.

Additional steps are required to set up a DSN for Windows users. `http://www.webwizguide.info/asp/tutorials/setting_up_dsn.asp` provides step by step instructions on how to make connections to an existing database through DSN.

3 Using SAGElyzer

`SAGElyzer` operates on potentially large data sets. Memory may be an issue for Windows operating system. Windows users are advised to (a) keep as few applicatons open as possible and (b) increase the memory allocated to R. One way to increase the memory

allocated to R is to right click the R icon and then click properties and append –max-mem-size=XXXM (XXX is the memory size desired that may vary between systems) to the end of the text in the entry box for *Target*. For example, suppose we have

`"C\Program Files\R\rw1080\bin\Rgui.exe"`

in the entry box for *Target*, the text will become

`"C\Program Files\R\rw1080\bin\Rgui.exe"--max-mem-size=512M`

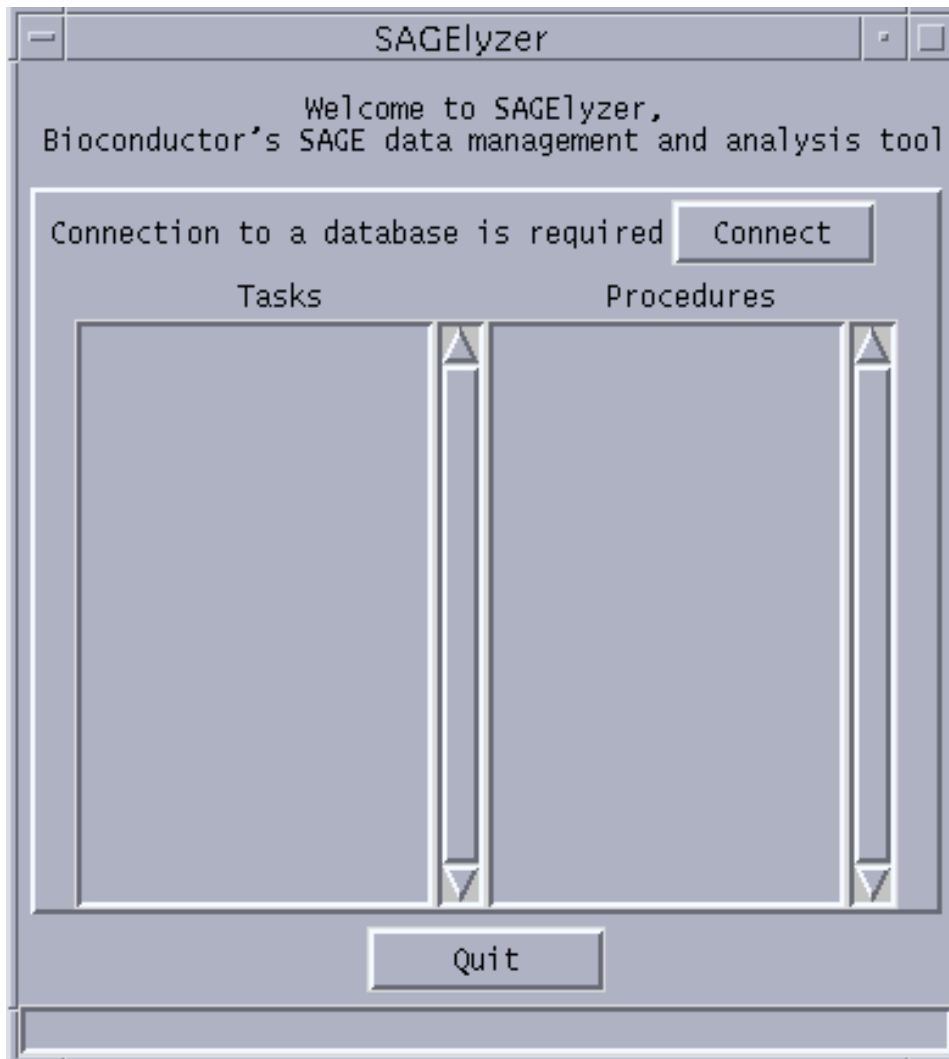if we would like to have 512M allocated to R.

Figure 1: A snapshot of the widget when `SAGElyzer` is loaded

When *SAGElyzer* is loaded by (checking for interactiveness is added to the code to turn off the code execution for automatic package building):

2

```
> if (interactive()) {
+     SAGElyzer()
+ }
```

A widget shown by Figure 1 will appear. The widget contains a *Connect* button in the upper right corner, two boxes in the middle, and a status bar at the bottom. As a connection to a database is always required, the *Connect* button is the only interactive feature at this moment. When the *Connect* button is clicked, a widget will pop up prompting users for inputs to make a connection to an existing database. The widget has different looks depending on the operating system. Figure 2 shows the widget for Unix. The one for Windows does not have the entry boxes for *Database, User, Password, or Host* but has an entry box for *DSN* instead. Entry boxes that are blank have to be filled by a user (if an argument is not required, e. g. password, the entry box for that argument can be left empty). Some of the entry boxes have already been filled with default values. Users may change the values for those entry boxes. However, if a user is not sure about what to enter, it is often safe to stay with the default values.
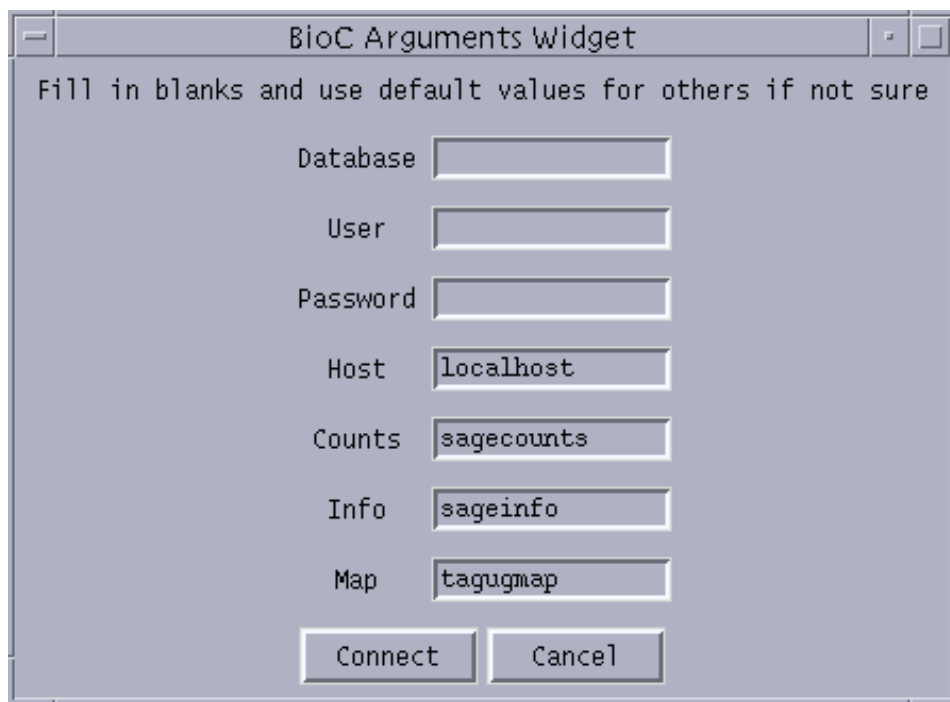
Figure 2: A snapshot of the widget for taking inputs for database connection

For both Windows and Unix systems, three database tables named by entries in the last three entry boxes (*Counts, Info, and Map*) will be created and maintained until updated later. The table for *Counts* contains counts of SAGE tags across libraries. The table for *Info* contains information about the orginal SAGE libraries and the database

table that stores counts for SAGE tags across libraries. The table for *Map* contains mappings between SAGE tags and UniGene ids.

After a connection to an existing database has been made, the tasks that can be perfomed using SAGElyzer will be made available to users through the *Tasks* box. Each of the buttons in the box is clickable and that in turn will have the procedures involved to perform the task listed as buttons in the *Procedures* box. Each button in the *Procedures* allows users to perform certain job related to the task. Figure 3 show the results when a database connection was made and the *Manage Data* button in *Tasks* box was clicked.
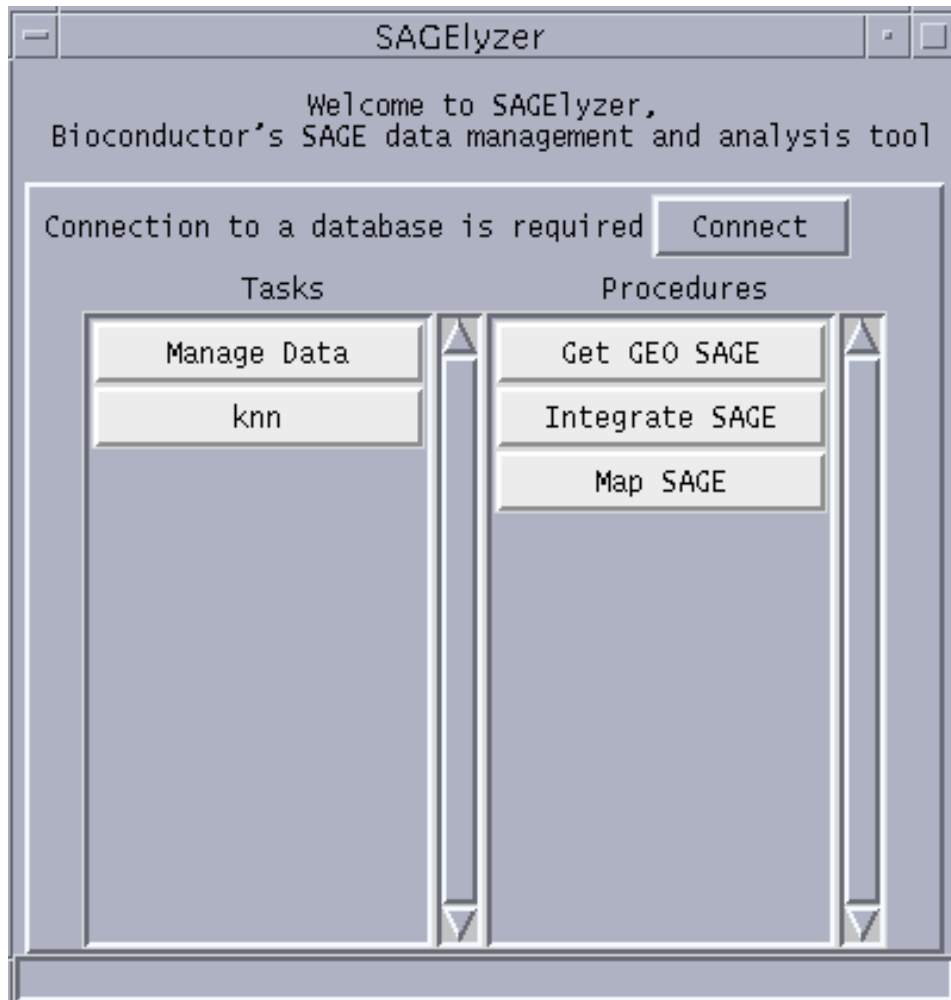

Figure 3: A snapshot of the widget when the *Manage Data* button in *Tasks* was clicked after a connection to an existing database had been made

If `SAGElyzer` is installed for the first time, users have to go through the procedures of *Manage Data* to have database tables created for later use. Below are descriptions of the three procedures of *Manage Data*:

Get GEO SAGE download all the SAGE libraries available at Gene Expression Omnibus (GEO. `http://www.ncbi.nlm.nih.gov/geo/`) for a given organism. The downloaded SAGE library data file will be saved in a local directory specified by a user and used to create database tables later.

Integrate SAGE integrate data from SAGE libraries stored locally and write the integrated data to database tables for later use. The procedure will populate the table for *Counts* and *Info*.

Map SAGE download data that map ASGE tags to UniGene ids and store the mappings in the database table for *Map* for later use.

Now, supose we would like to get SAGE libraries for human from *GEO* and click *Get GEO SAGE*. We will see a resulting widget (Figure 4) prompting for inputs for the name of the organism (*Organism*) we are interested in, a name for an existing directory to save the downloaded SAGE libraries to (*Save To*), and the URL from which SAGE libraries are available (*Source URL*). The default value for *Source URL* was correct at the time of the writing.
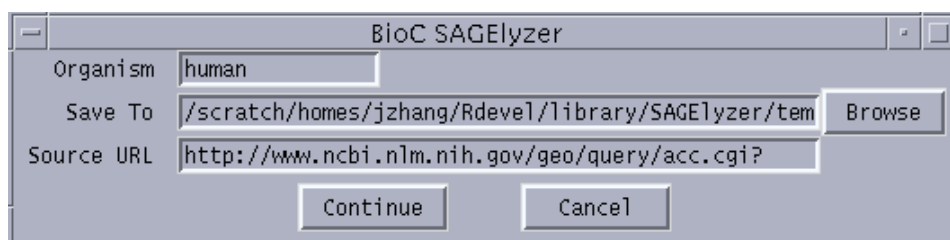

Figure 4: A snapshot of the widget when the *Get GEO SAGE* button in *Procedures* was clicked

If we stay with the default and click *Continue*, it will take a while for the procedure to finish because there are quite a few files to be downloaded from the source. SAGE library files downloaded will have an extension ".sage". If you do not want to wait, you may skip this procedure as we have two sample SAGE data files stored in the temp directory (with an extension ".test") for you to use for now.

When we have SAGE libraries (downloaded or created) stored in a local directory, we can invoke *Integrate SAGE* to integrate the data and write them to database tables. Clicking *Integrate SAGE* invokes another widget (Figure 5) that taks inputs for 4 arguments. The input for *Library* can be a file name or the name for a directory containing SAGE library data files. The radio buttons for *Directory* have to be set to TRUE if *Library* is the name for a directory or FALSE for a file. The value for *Skip* determines how many rows to skip from the top of each of the SAGE library data files when the `SAGElyzer` reads data from files and that for *Pattern* tells `SAGElyzer` only to process data files with certain extension or pattens that match the one defined by a user.
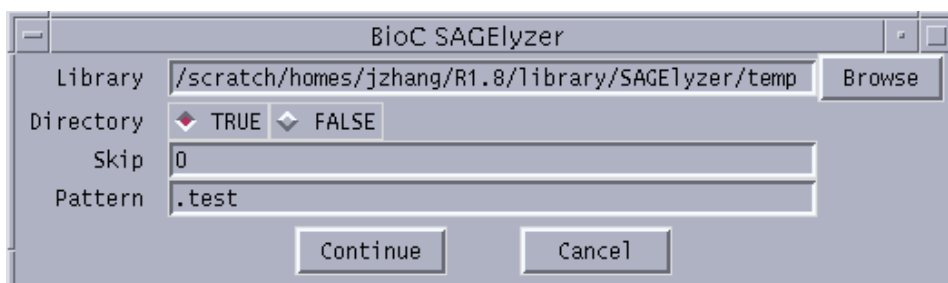
5

Figure 5: A snapshot of the widget when the *Integrate SAGE* button in *Procedures* was clicked

In our example, we have saved the downloaded (or stored) SAGE library data from *GEO* to the library defined by the defulat value for *Library*. As SAGE library data from *GEO* have no headers, we set the value for *Skip* to 0 and set *Directory* to TRUE. If you have downloaded SAGE library files from *GEO* and willing to wait, you stay with the default value for *Pattern* and click *Continue*. Otherwise, change the value for *Pattern* from ".sage" to ".test" to only process the two test sample files and click *Continue*. the system will integrate the data and write them to the database. Again, it will take a while for the procedure to finish if you choose to process data files downloaded from *GEO*. The status bar at the bottom of the main `SAGElyzer` widget wil have something reads "Running procedure 'XXX'. Please wait." when any of the procedures is running.

Integrate SAGE requires that all the data files to be processed should be stored in the same directory, either with or without a header, and have the same pattern for their names. These usually may not be a problem when only SAGE data that are downloaded from *GEO* will be used. If both dowloaded and local data will be used, efforts have to be made to satisfy the requirements before integrating the data.

Now, we have the SAGE libraries downloaded from *GEO* written to the database. Another thing we need to do is to get the mappings from SAGE tags to UniGene ids. If we click *Map SAGE*, we will be prompted by another widget (Figure 6) for three arguments. *DB Table Name* is for the name of the database table where the mappings will be stored. *Map* indicates whether the mapping are from SAGE tag to UniGene ids or UniGene id to SAGE tag. *Source URL* is the URL for the source file (SAGEmap_tag_ug-rel.zip) containing mappings between SAGE tags and UniGene ids. In our example, we stay with the defaults and click *Continue*. The procedure will finish after a while.

Task *Manage Data* only needs to be performed when `SAGElyzer` is first installed or the existing database tables need to be updated. The database tables created will be available for later use any time when `SAGElyzer` is loaded thereafter.

Once the database tables have been created in a previous session, we can invoke the procedures that relies on the database tables for analyse or annotation. If we click *knn* in *Tasks*, the *Procedures* box will be populated with 5 buttons with the fillowing functionalities:
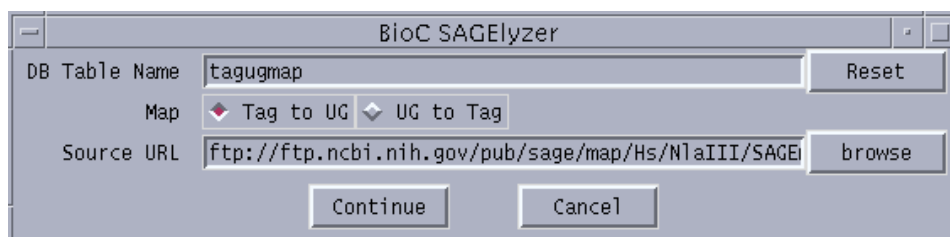
Figure 6: A snapshot of the widget when the *Map SAGE* button in *Procedures* was clicked

Set arguments set values for arguments that are reguired to perform knn.

Run knn permorm knn using arguments set by *Set arguments*.

Get counts get counts for tags found by *knn* across selected SAGE libraries.

Map SAGE map the tags found by *knn* to UniGene ids and link the UniGene ids to UniGene web site.

Find neighbor genes find genes neighboring tags found by *knn* within a range up and down streams of a chromosome. This procedure requires that a data package *XXXCHRLOC* (available from Bioconductor) be installed, where XXX is the name of the organism of interest (i. g. "human")

Set Arguments and *Run knn* have to be run in sequence before other procedures can show any useful results. Lets run *Set Arguments* first. The arguments needed for knn will be listed by a resulting widget shown by Figure 7.

The widget has an entry box for *Target tag* for which a given number of tags (defined by *k value*) that have a similar pattern of expression will be found across the libraries selected by a user. `SAGElyzer` currently only takes a ten-letter tag for NlaII. If a users does not select any library using the widget, all the libraries will be used to identify the tags with a similar pattern of expression. Values for *Normalization, Distance, k value, and Transformation* can be chosen from a set of options. Lets type in "aaaaaaaaaa" for *Target tag*, select two libraries, set *k value* to 50, keep the default values for the others, and click *Continue*.

When the arguments have been set, we can click *Run knn* to find similarly expressed tags. Depending on how big a data set will be involved, the procedure will take various lenths of time to finish. The result of the execution will be a widget showing a list of tags and the calculated distances. A user have the option to save or not save the result.

Users are also allowed to view the counts for tags identified by *Run knn* across the selected libraries the *Get Counts*. Again, the result will be widget that lists the tags and their counts across the selected libraries. The procedure may take a while to finish when the database table is large or a large number of SAGE libraries have been selected.
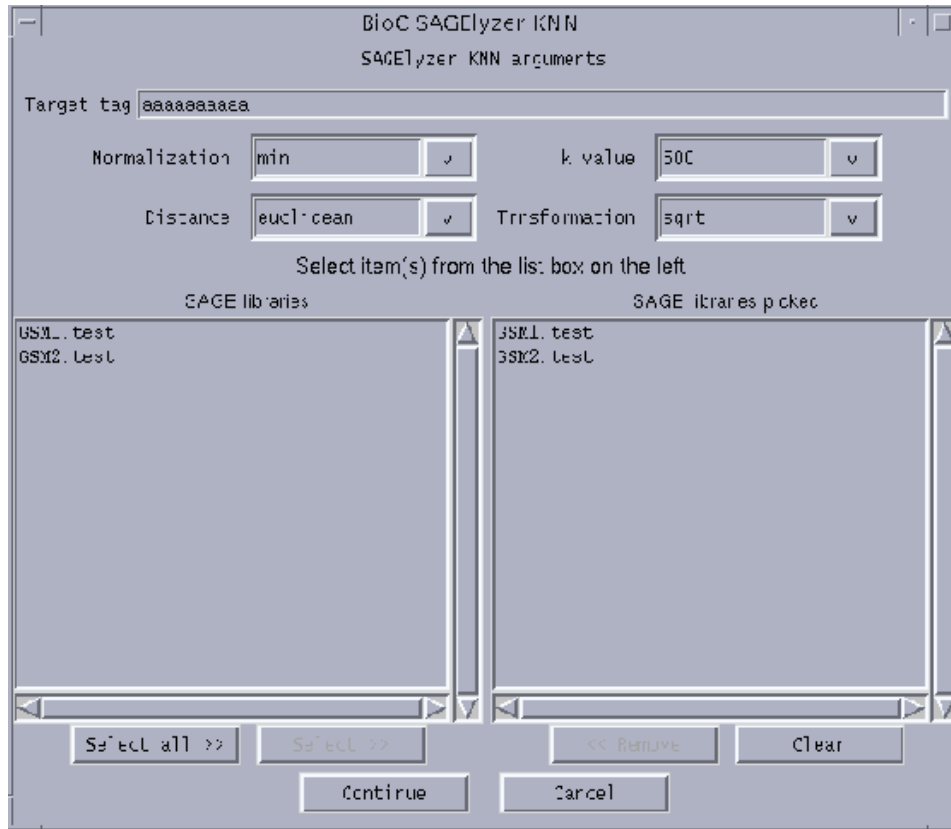
7

Figure 7: A snapshot of the widget when the *Set Arguments* button in *Procedures* was clicked. The snapshot was from an example using the two test SAGE data files.

Map SAGE generates an HTML file that maps the tags identified tags to UniGene ids. Clicking the UniGene id mapped to any of the tags will bring the user to UniGene web site where information about the UniGene cluster is available.

Find neighbor genes invokes a widget with the tags identified by *Run knn* mapped to UniGene ids. The mappings are listed in a set of list boxes. Clinking any of the entries in the list boxes will show the LocusLink ids of genes that are within a specified range up and down stream from the tag along the chromosome the tag is located.