Network Biology

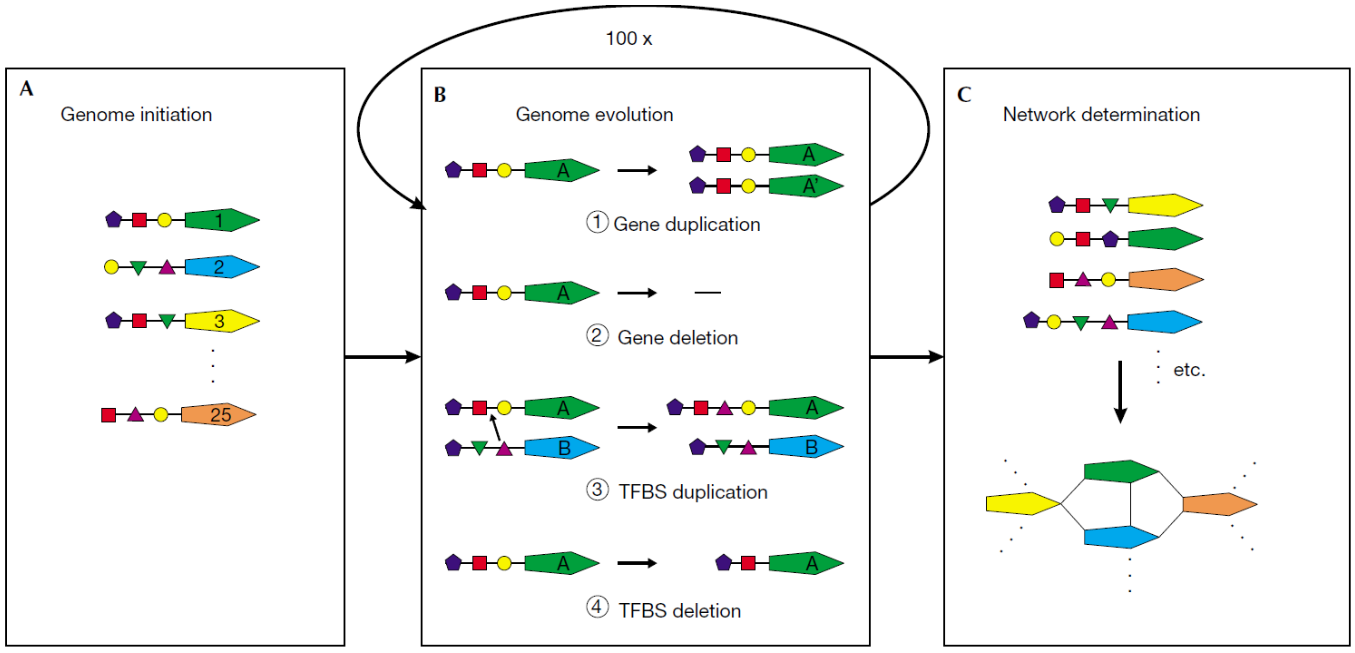
Coding Assignment 2

Note: You are responsible for the backup of data as well as results, which will be used for evaluation.

1. Implement the "evolutionary model of transcription regulation".
   i. Start with an initial pool of small number of genes (say, 25) and transcription factor binding sites (say, 10) such that each gene has a fixed number of TFBS (say, 3).
   ii. Implement genome evolution involving gene duplication, gene deletion, TFBS duplication, and TFBS deletion. Play with the probability values for each of these events.
   iii. After a certain number of steps of evolution (say, 100), construct the co-expression network with the assumption that 'two genes co-express if they share one or more TFBS'.

   For specific initial values of number of genes and TFBS, probabilities used in the evolution step, and total number of steps in evolution: **[10+10+5]**

   a. Report the **final co-expression network** (image exported from Cytoscape).
   b. Plot the **degree distribution** of the networks for following cases: Gene Duplication ≪ TFBS Duplication, TFBS Duplication ≫ Gene Duplication, and Gene Duplication ≈ TFBS Duplication.
   c. Provide your **conclusions** based on these results.



('The yeast coexpression network has a small-world, scale-free architecture and can be explained by a simple model', V van Noort, B Snel & M Huynen, *EMBO Rep.* 2004; 5(3): 274–279.)

2. Create Residue Interaction Graph (RIG) and Long-Range Interaction Network (LIN) models for ***any five* of the following thirty single-domain two-state folding proteins**, using the strategy listed below: 1hrc, 1imq, 1ycc, 2abd, 2pdd, 1aps, 1cis, 1coa, 1fkb, 1hdn, 1pba, 1ubq, 1urn, 1vik, 2hqi, 2ptl, 2vik, 1aey, 1csp, 1mjc, 1nyf, 1pks, 1shf, 1shg, 1srl, 1ten, 1tit, 1wit, 2ait, 3mef. [**10**]
    - Download the PDB file.
    - Extract the atomic coordinates information (in any programming language, for extracting the coordinates flawlessly).
    - Further, extract the coordinates of Cα atoms of each amino acid.
    - Using the data of representative Cα atoms, **write a code** to create the RIG model using a cut-off of 7.
    - **Write a code** to compute the LIN model, using the threshold of 12 amino acids.

    a. Compute the characteristic path length (L) and clustering coefficient (C) of both RIG and LIN models for each of the above proteins. **Provide your results in a tabular form**.
    b. Write your observations about their topological properties and expected rate of folding.

3. Implement Bartoli's model of protein structure for the five proteins that you chose in response to Question 2. Compute the characteristic path length (L) and clustering coefficient (C) for the RIG model equivalent of Bartoli's models (for 100 instances).

    Following is the procedure for generating contact maps using Bartoli's model:
    (i) Assign 1s to the first two diagonals (up and down the main diagonal) of the adjacency matrix in order to define the backbone contacts.
    (ii) Randomly select a pair of residues *i* and *j* with a probability that decreases linearly with the distance separating these residues in the protein sequence.
    (iii) Assign 1s to the entries of the adjacency matrix corresponding to all nine residue pairs generated by the Cartesian product of $\{i-1, i, i+1\} \times \{j-1, j, j+1\}$.
    (iv) Iterate the last procedure until the number of links in the random graph is *close to* those of the real protein.

    a. **Compare the results** of the RIG and LIN of the original structure to that of their Bartoli counterparts, **and write your observations**. Can you modify step (ii) to create contact maps that are closer to those from real protein structures? Report your results. [**10**]
    b. Compute the plot depicting the 'number of amino acid contacts made' and 'Cartesian distance between them' for all five proteins. Could it be possible to create a model to generate a contact map of a protein, that accounts for this feature? [**5**]

GitHub

GIT CHEAT SHEET

Git is the free and open source distributed version control system that's responsible for everything GitHub related that happens locally on your computer. This cheat sheet features the most important and commonly used Git commands for easy reference.

INSTALLATION & GUIS

With platform specific installers for Git, GitHub also provides the ease of staying up-to-date with the latest releases of the command line tool while providing a graphical user interface for day-to-day interaction, review, and repository synchronization.

GitHub for Windows
https://windows.github.com

GitHub for Mac
https://mac.github.com

For Linux and Solaris platforms, the latest release is available on the official Git web site.

Git for All Platforms
http://git-scm.com

SETUP

Configuring user information used across all local repositories

```
git config --global user.name “[firstname lastname]”
```

set a name that is identifiable for credit when review version history

```
git config --global user.email “[valid-email]”
```

set an email address that will be associated with each history marker

```
git config --global color.ui auto
```

set automatic command line coloring for Git for easy reviewing

SETUP & INIT

Configuring user information, initializing and cloning repositories

```
git init
```

initialize an existing directory as a Git repository

```
git clone [url]
```

retrieve an entire repository from a hosted location via URL

STAGE & SNAPSHOT

Working with snapshots and the Git staging area

```
git status
```

show modified files in working directory, staged for your next commit

```
git add [file]
```

add a file as it looks now to your next commit (stage)

```
git reset [file]
```

unstage a file while retaining the changes in working directory

```
git diff
```

diff of what is changed but not staged

```
git diff --staged
```

diff of what is staged but not yet committed

```
git commit -m “[descriptive message]”
```

commit your staged content as a new commit snapshot

BRANCH & MERGE

Isolating work in branches, changing context, and integrating changes

```
git branch
```

list your branches. a * will appear next to the currently active branch

```
git branch [branch-name]
```

create a new branch at the current commit

```
git checkout
```

switch to another branch and check it out into your working directory

```
git merge [branch]
```

merge the specified branch’s history into the current one

```
git log
```

show all commits in the current branch’s history

INSPECT & COMPARE

Examining logs, diffs and object information

`git log`
show the commit history for the currently active branch
`git log branchB..branchA`
show the commits on branchA that are not on branchB
`git log --follow [file]`
show the commits that changed file, even across renames
`git diff branchB...branchA`
show the diff of what is in branchA that is not in branchB
`git show [SHA]`
show any object in Git in human-readable format

TRACKING PATH CHANGES

Versioning file removes and path changes

`git rm [file]`
delete the file from project and stage the removal for commit
`git mv [existing-path] [new-path]`
change an existing file path and stage the move
`git log --stat -M`
show all commit logs with indication of any paths that moved

IGNORING PATTERNS

Preventing unintentional staging or commiting of files

```
logs/
*.notes
pattern*/
```

Save a file with desired patterns as .gitignore with either direct string matches or wildcard globs.

`git config --global core.excludesfile [file]`
system wide ignore pattern for all local repositories

SHARE & UPDATE

Retrieving updates from another repository and updating local repos

`git remote add [alias] [url]`
add a git URL as an alias
`git fetch [alias]`
fetch down all the branches from that Git remote
`git merge [alias]/[branch]`
merge a remote branch into your current branch to bring it up to date
`git push [alias] [branch]`
Transmit local branch commits to the remote repository branch
`git pull`
fetch and merge any commits from the tracking remote branch

REWRITE HISTORY

Rewriting branches, updating commits and clearing history

`git rebase [branch]`
apply any commits of current branch ahead of specified one
`git reset --hard [commit]`
clear staging area, rewrite working tree from specified commit

TEMPORARY COMMITS

Temporarily store modified, tracked files in order to change branches

`git stash`
Save modified and staged changes
`git stash list`
list stack-order of stashed file changes
`git stash pop`
write working from top of stash stack
`git stash drop`
discard the changes from top of stash stack