Bioinformatics III

Prof. Dr. Volkhard Helms
Maryam Nazarieh, Duy Nguyen, Thorsten Will
Winter Semester 2016/2017

Saarland University
Chair for Computational Biology

Exercise Sheet 2

Due: Nov 11, 2016 13:15

Submit your solutions on paper, hand-written or printed at the *beginning* of the lecture or in building E2.1, Room 3.02. Alternatively you may send an email with a single PDF attachment. If possible, please include source code listings. Additionally hand in all source code via mail to thorsten.will@bioinformatik.uni-saarland.de.

2 Force directed layouts and real interaction networks

We continue to evolve the classes from the first assignment. The assignment of this week deals with energies and forces applied to layout networks and real data on protein-protein interaction networks.

Exercise 2.1: Force directed layout of graphs (50 pts)

In this exercise you implement a layout algorithm for networks in the **Layout**-class by using energy functions that mimic repulsive and attractive molecular forces. Subsequently, you read networks from files and visualize the final layouts and the energy trajectories.

General remarks:

- The force equals the negative gradient of the energy, i.e., the force is a measure for how much the energy changes with an infinitesimal displacement:

$$\vec{F}(\vec{r}) = -\nabla E(\vec{r}), \text{ with the gradient operator } \nabla := \begin{pmatrix} \mathrm{d}/\mathrm{d}x \\ \mathrm{d}/\mathrm{d}y \\ \mathrm{d}/\mathrm{d}z \end{pmatrix}$$

  In a single dimension, this reduces to $\nabla = \mathrm{d}/\mathrm{d}r$, i.e. the simple derivative with respect to the distance r. The gradient of a function can consequently be understood as a multidimensional slope.

- The interaction energy between two charges q_1 and q_2 is given as:

$$E_c(r) = \frac{1}{4\pi \cdot \epsilon_0 \epsilon} \frac{q_1 q_2}{r}$$

  For the connecting spring use the harmonic potential:

$$E_h(r) = \frac{kr^2}{2}$$

(a) As a preliminary consideration, use the general definitions above to calculate the force fields $\vec{F}(\vec{r}) = -\nabla E(\vec{r})$ for both the Coulomb interaction E_c and the harmonic potential E_h in cartesian coordinates.
Write ∇ and the resulting force field $\vec{F}(\vec{r})$ in component form to get one equation for x, y, and z, each. This is the form that you need to implement the layout algorithm in the next exercise. Note that:

$$r = \sqrt{x^2 + y^2 + z^2}, \qquad \vec{F}(\vec{r}) = \begin{pmatrix} F_x(x) \\ F_y(y) \\ F_z(z) \end{pmatrix}$$

(b) For the implementation, use variants of such interaction energies that are adapted to graphs. Between all nodes, use a repulsive degree dependent Coulomb type potential, defined as:

$$E_c(r_{ij}) = \frac{k_i \cdot k_j}{r_{ij}}$$

Additionally, for interacting nodes, use a degree independent harmonic attractive potential:

$$E_h(r_{ij}) = \frac{r_{ij}^2}{2}$$

The parameter r_{ij} is the distance between two nodes i and j. Because we layout in 2D, the squared distance is defined as:

$$r_{ij}^2 = (x_i - x_j)^2 + (y_i - y_j)^2$$

The interaction between two nodes i, j is defined as:

$$W[i][j] = \begin{cases} 1, & \text{if edge } i \to j \text{ exists} \\ 0, & \text{else} \end{cases}$$

The basic approach (function **layout(iterations)**) can be outlined as:

(1) Calculate the pairwise forces between all nodes and sum them up for each of the nodes:

$$\vec{F}_{ij} = \vec{F}_c(\vec{r}_{ij}) + W[i][j] \cdot \vec{F}_h(\vec{r}_{ij})$$

Thus, the total force on node i is $F_i = \sum_j F_{ij}$. Note that the forces between two nodes are symmetric, i.e., $F_{ij} = -F_{ji}$.

(2) Update the position of each node from the forces as:

$$\Delta r_i = \alpha \cdot F_i$$

A reasonable value is $\alpha = 0.03$. Do not forget to reset all the forces after this step.

(3) Calculate the total energy, which is the sum of all individual interaction energies:

$$E_{\text{tot}} = \sum_{j>i} E_c(r_{ij}) + W[i][j] \cdot E_h(r_{ij})$$

The energy of each iteration is stored and returned, the positions are altered in the Node-objects.

The alternative function **SAlayout(iterations)** additionally adds a random force, a "thermal contribution", to the total force on each node in each iteration which should decrease (implement!) in every step. This optimization principle is called simulated annealing. Why is it worthwhile in practice?
To store energy and positions the Node-class is extended *"on-the-fly"* (see source code). This will magically add those attributes to your Node-objects .

(c) Implement **GenericNetwork**, a network class that imports networks from files.

(d) Use the new classes to layout the test files "`star.txt`", "`square.txt`", "`star++.txt`" and "`dog.txt`", which are part of the supplement. Do 1000 iterations with both implementations of the algorithm, report the final energies and plot the nicer layout. For one of the networks, also compare the energies per step for the basic and the simulated annealing method. **Tools.py** contains new methods that you can use for plotting. You may need to restrict to certain ranges of the axes to see the important differences.

(e) A general approach to speed up graph algorithms is to merge sets of nodes into "supernodes" that represent them. Briefly describe how a layout procedure that uses this idea could look like (in words, at most some lines of pseudecode, usage of common graph algorithms allowed). Argument why the runtime improves and why you think your solution is reasonable in practice.

Exercise 2.2: Real interaction networks (50 pts)

BioGRID ("Biological General Repository for Interaction Datasets") is a protein interaction database which, in version 3.4.142 (Nov. 2015), contains data of 842,529 raw protein and genetic interactions from major model organism species compiled from 57,513 publications. The supplement contains this release as a tab-separated file ("`BioGRID.txt`"). The format is documented in the beginning of the file, make yourself familiar with that.

In this exercise you implement the class **BioGRIDReader** which should help you to deal with such data.

(a) The class should read the file in its initialization and store the necessary data in a data structure that simplifies your later queries. For every organism found in the file (as **NCBI taxon identifiers**) one should be able to retrieve all interactions as pairs of **official gene symbols** easily.

(b) Implement **getMostAbundantTaxonIDs(n)** and use it to return the **five** organism with the most interactions annotated in BioGRID as well as their respective number of interactions. Argument why the order is not surprising.

(c) How big is the human interaction network and which are the **10** proteins with the highest degree? Take one of them as an example and briefly explain the biology behind the connectivity.

(d) Implement **writeInteractionFile(taxon_id, filename)** to be able to create organism-specific network files that can be used by the GenericNetwork-class. Build a network for human (taxon 9606), determine and plot the corresponding degree distribution. Discuss if the distribution behaves more like a scale-free or a random network.

(e) At last, analytically assess a part of the the previous issue. The degree distribution of a scale-free network follows a power law, which has the form

$$P(k) \sim k^{-\gamma}.$$

To simplify the exercise, we assume $P(k) = Ck^{-\gamma}$, with C being a fixed normalization constant to obtain a proper distribution. Now fit this theoretical distribution to the degree distribution of the human interaction network using the Kolmogorov-Smirnov (KS) distance. Follow this guideline:

- Implement **Tools.getScaleFreeDistributionHistogram(gamma, k)** which returns such a simple power law distribution ($\text{histogram[i]} = \text{math.}\textbf{pow}(\text{i}, -\text{gamma})$ and normalization afterwards).

- Implement the KS distance in **Tools.simpleKSdist(histogram_a, histogram_b)**: The KS distance of two distributions is the maximal distance between their respective **cumulative** distributions F_i:

  $$D = \sup_x |F_1(x) - F_2(x)|$$

  Thus, first build cumulative distributions from the normalized histograms, then find the position where the distributions deviate the most and return this distance.
- Use the KS distance to determine a γ (between 1 and 2, 0.1 steps sufficient) for the power law distribution that fits best to the degree distribution of the human interaction network. Compare the empirical distribution of the network to the theoretical distributions in a double-log. plot. Comment on the quality of your fit, reason why it may fail and how it could be vastly improved.
  You can use **compareToTheory.py** as a template to program this study.

Have fun!