Bioinformatics III

Prof. Dr. Volkhard Helms — Saarland University
Ruslan Akulenko, Maryam Nazarieh, Duy Nguyen, Thorsten Will — Chair for Computational Biology
Winter Semester 2014/2015

Exercise Sheet 2

Due: November 7, 2014 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 More On Networks, Forces, and Force Directed Layouts

We continue to evolve our classes from the first assignment. The assignment of this week first deals with the construction of scale-free networks, then we will discuss energies and forces to finally be able to layout networks.

Exercise 2.1: The Scale-Free Network (35 pts)

First, (a) construct a scale-free network according to the Barabási-Albert model. Then (b) examine the degree distribution of such networks and determine some characteristics in comparison to random networks. Finally, in (c) try to fit the degree distribution to a theoretical distribution.

(a) Implement the algorithm given in the lecture to set up a scale-free network according to the Barabási-Albert model. Start from the first three connected nodes and add each new node with a given number of links. Connect the new links with increasing preference to nodes that have higher degrees. This **ScaleFreeNetwork**-class should again use the abstract network class that you wrote in the first assignment.
To obtain a much faster implementation and full points, think of a method to map the probabilities to connect to nodes somehow instead of computing them from scratch in each iteration.

(b) Determine the degree distributions for scale-free networks of 10 000 and 100 000 nodes (each with two new links per iteration), respectively, and plot them with double logarithmic axes. A new pre-implemented method in **Tools.py** will help you with that. What are the differences?
Next, compare one of the distributions to the degree distribution of an equally sized random network (play around with the plot-scaling). What are the major differences?

(c) 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. Try to fit this theoretical distribution to the degree distribution of a random network using the Kolmogorov-Smirnov distance. Follow this guideline:

- Implement **Tools.getScaleFreeDistributionHistogram(gamma, k)** which returns such a simple power law distribution (histogram[i] = math.**pow**(i, −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 3, 0.1 steps sufficient) that fits best to the degree distribution of a scale-free network with 10 000 nodes and two new links per iteration. Compare the empirical distribution of the network to the theoretical distribution with optimal γ in a double-log. plot. Comment on the quality of your fit, reason why it may fail and how it could be vastly improved.

Exercise 2.2: Energy and forces (20 pts)

(a) **Configuration of minimal energy:**
Determine the equilibrium distance between two equally charged mass points which are connected by a spring. At the equilibrium distance the total force vanishes. Verify that instead of calculating the forces explicitly, it is equivalent to determine the configuration of minimal energy.

Hints:

- 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:

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

- To show the equivalence of vanishing force and minimal energy remember how the minimum of a function is defined. Also note that the distance between two particles is a one-dimensional measure.

(b) **Force field from a spherically symmetric potential:**
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.

Hints:

- 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:

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

Exercise 2.3: Force directed layout of graphs (45 pts)

Implement a layout algorithm algorithm for your networks in the **Layout**-class by using energy functions that mimic the repulsive and attractive behavior as in the previous exercise. Subsequently, read networks from files and visualize the final layouts and the energy trajectories.

(a) 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:

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

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., $F_{ij} = -F_{ij}$.

(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?

Hint:

- 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 .

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

(c) 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.

Have fun!