Final Examination
COT 5405

Due December 10, 2012 between 1:30pm and 2:30pm in the class.

This test became available on December 3, 2012 at 1:30pm.

Solve any one of the following problems. If you solve more than one problem, only the first two solutions will be evaluated and the maximum of the two scores will be used. It is recommended that you solve only one problem.

If you prefer to take an in-class closed-book examination, please register your intent to do so with the instructor before 3:30pm on December 3, 2012. No Late Registrations!

Do not discuss these problems with anyone else. You are free to use notes, books, papers, libraries, and the internet. You are also free to use software libraries and any existing source code that you find useful. **You must cite all references used. Plagiarism will be reported and will result in a Fail grade for this class.**

Problem 1: Structured "Big-Data" and Graphs

Flow cytometry is an experimental technique that measures a fixed number of parameters of every cell in a biological sample. Consider a biological sample with n cells and a flow cytometer that can measure m parameters of every cell in this sample. Thus, the flow cytometer produces a 2-dimensional array T with n rows and m columns. Each entry $T(i,j) \in \mathbb{R}$ denotes the measured value of the j^{th} parameter for the i^{th} cell in the sample ($1 \leq i \leq n, 1 \leq j \leq m$). We will use $T(i,:)$ to denote the vector representing the i^{th} row of the 2-dimensional array, and $T(:,j)$ to denote the vector representing the j^{th} column of the 2-dimensional array T. It is known that $10^4 \leq n \leq 10^6$ and $10^1 \leq m \leq 10^2$.

1. Flow cytometry is an error-prone measurement process. In particular, we are interested in removing any "outliers" in the data.

   (a) A vector v is said to be *supported* by a vector w if and only if (i) $||v - w|| > 0$, and (ii) $\frac{||v-w||}{max(||v||,||w||)} < C_n$.

   (b) Measurement on the i^{th} cell i.e. $T(i,:)$ is said to be *suspect* if and only if the row $T(i,:)$ is *supported* by no more than $\frac{n}{1000}$ data points.

   Determine a value of C_n such that the percentage of *suspect* measurements is between 0.01% and 20%. Compute all the cells whose measurements are suspect (in each of the given two-dimensional arrays).

   (a) For each 2-dimensional array, plot the original data values in blue and highlight the *suspect* measurements in red.

   (b) Construct a table indicating the percentage of measurements that has been classified as "suspect" for each of the provided 2-dimensional arrays.

2. Consider all the measurements in the 2-dimensional array T that are not *suspect* using the value of C_n determined earlier. Let us represent them using the 2-dimensional array U with u ($\leq n$) rows and m columns.

   The graph G_U is the graph representing these data values. In particular, $G_U = (V_U, E_U)$, where $v_i \in V_U$ represents the measurement corresponding to the i^{th} cell whose measurement is not suspect or the row $U(i,:)$ in the 2-dimensional array U.

   Let $D = \{d_{ij}\} = \{||v_i - v_j|| \mid 1 \leq i \leq u, 1 \leq j \leq u\}$. Compute a number d such that d is smaller than at least 90% of all the elements in the set D and it is larger than at least 1% of all the elements in the set D. Then, $(v_i, v_j) \in E_U$ if and only if $||U(i,:) - U(j,:)|| \leq d$.

   (a) Compute and store the adjacency list representation of the graph G_U

   (b) Compute the minimum spanning tree T_{min} of G_U. Each edge (v_i, v_j) in G_U is assigned the weight $||v_i - v_j||$.

(c) Let $W_1, W_2, \ldots W_{u-1}$ denote the edges of the MST T_{min} in descending order of their weight. Compute the top 100 edges $W_1, W_2, \ldots, W_{100}$ in the minimum spanning tree T_{min}.

(d) How many clusters are present in G_U?

3. In the paper by Cami and Deo, we studied a preferential attachment and deletion model for studying web-like graphs.

   (a) Show that G_U is (likely, or) not likely to be a graph produced by preferential attachment and deletion. Present a clear and mathematical argument.

   (b) Construct a graph evolution model $G_t = (V_t, E_t)$ with two operators *Clone* and *Differentiate*. A graph evolves using the following rules:

      i. *Clone*: A "clone" node is added to the graph with probability p_c. A node v is added by cloning a *randomly chosen* (with uniform probability) existing node u. All the edges $(u, w) \in E_t$ incident on the node u are also cloned by adding a new edge (v, w) for every existing edge (u, w). An edge is also added between the newly added node v and the node u that was cloned.

      ii. *Differentiate*: A *differentiated* node is added to the graph with probability p_d. A differentiated node v is added by creating a new node. No edges connect this node to any existing node in the network.

      Compute the expected values of the number of nodes, the number of edges, and the edge density of the evolving graph G_t. You may also compute any other topological properties that you like (such as number of connected components).

On the class website, ten (10) 2-dimensional arrays or tables have been provided for use with this assignment.

Problem 2: Algorithm for Controlling Networks

The Cami-Deo model presented in the class is an example of a dynamic model for understanding complex networks. In general, a model M describes a set of n topological properties $T_i (1 \leq i \leq n)$ of an evolving graph, such as edges, nodes, edge density, and number of connected components. For each i, the model describes the evolution of T_i. We may assume that the topological property T_i at time $t+1$ only depends on the topological properties at time t, i.e.,

$$T_i(t+1) = T_i(t) + f_i(T_1(t), T_2(t), \dots, T_n(t)) \tag{1}$$

Note that $f_i (1 \leq i \leq n)$ are known (given) functions.

Assume that we are able to influence a subset of k topological properties of the graph using an external time-varying input $u_j(t) (1 \leq j \leq k \leq n)$ that we can directly control. Thus, the complete model is given by the following recurrence equations:

$$\begin{aligned}
T_1(t+1) &= T_1(t) + f_1(T_1(t), T_2(t), \dots, T_n(t)) \quad + \quad u_1(t) && (2)\\
T_2(t+1) &= T_2(t) + f_2(T_1(t), T_2(t), \dots, T_n(t)) \quad + \quad u_2(t) && (3)\\
&\dots\\
T_k(t+1) &= T_k(t) + f_k(T_1(t), T_2(t), \dots, T_n(t)) \quad + \quad u_k(t) && (4)\\
T_{k+1}(t+1) &= T_{k+1}(t) + f_{k+1}(T_1(t), T_2(t), \dots, T_n(t)) && (5)\\
&\dots\\
T_n(t+1) &= T_n(t) + f_n(T_1(t), T_2(t), \dots, T_n(t)) && (6)
\end{aligned}$$

Given $T_1(0), T_2(0), \dots T_n(0)$, the functions $f_1, f_2, \dots f_n$, and an objective function that we seek to minimize:

$$g(T_1(t_f), T_2(t_f), \dots, T_n(t_f)) + \sum_{t=0}^{t_f} h(u_1(t), u_2(t), \dots u_k(t)) \tag{7}$$

Note that the functions g and h are known (given). Suggest an algorithm for computing the inputs $u_1, u_2 \dots u_k$ that minimizes the objective function. Analyze the run-time of the algorithm. Implement it on a network model of your choice.

Problem 3: Risk Optimization

Given n assets A_i with *known* prices $S_i \in \mathbb{N}$ $(1 \leq i \leq n)$, we seek to synthesize the corresponding quantities $\hat{n}_i \in \mathbb{N}$ of assets that should be held in the portfolio Π. In general, the value of the portfolio is given by

$$\Pi = \sum_{i=1}^{n} \hat{n}_i S_i \tag{8}$$

Goal: Synthesize a portfolio Π i.e. determine the quantity $n_i \in \mathbb{N}$ $(1 \leq i \leq n)$ of every asset A_i, such that

1. The contribution of each asset A_i to the overall risk of the portfolio is the same, i.e., for all i, j $(1 \leq i \leq n, 1 \leq j \leq n)$,

$$\hat{n}_i S_i \sigma_i \sum_{k=1}^{n} (\hat{n}_k S_k \sigma_k \rho_{ik}) = \hat{n}_j S_j \sigma_j \sum_{k=1}^{n} (\hat{n}_k S_k \sigma_k \rho_{jk}) \tag{9}$$

Note that $\sigma_i \in \mathbb{N}$ denotes the known risk contribution from the asset A_i and $\rho_{ij} \in \mathbb{N}$ denotes a known measure of the correlation of the risk between assets A_i and A_j. The vector of risk measures $\sigma_1, \ldots, \sigma_n$ and the matrix of correlation measures between the risks $\rho_{11}, \rho_{12}, \ldots, \rho_{nn}$ are known *apriori*.

- State the decision version of this problem. Is it in NP?
- Is the decision version of this problem NP-complete? Present a formal argument.
- Design an algorithm for solving this problem? Compute the runtime complexity of your algorithm. Using randomized inputs, compute the size of the largest problem that your algorithm/implementation can solve.

Problem 4: Edit Distance Between Structured Data

Given a weighted tree $T_1 = (V_1,\ E_1)$ and an unweighted tree $T_2 = (V_2,\ E_2)$, we want to compute the distance $\mathcal{D}(T_1,\ T_2)$ between the two trees:

1. **Assumption**: You may assume that both T_1 and T_2 are directed and rooted trees, and that every node has no more than $k-$children (for some small constant k).
2. **Assumption**: Both T_1 and T_2 have the same height. All the leaves have the same depth.
3. **Assumption**: A path in a tree must start at the root and end at one of the leaves.
4. Given a path ρ_1 in the tree T_1, let $N(\rho_1)$ be the nearest path in the tree T_2, where distance between two paths is computed as the edit distance between those paths (identical to the edit distance between strings). The edit distance between two paths $D(\rho_1, \rho_2)$ is the edit distance between the two strings representing the nodes along the two paths.
5. The weight $W(\rho_1)$ of a path ρ_1 is the produce of the weights on the edges along the path ρ_1.
6. The distance $\mathcal{D}(\rho_1, T_2)$ between a path ρ_1 and an unwieghted tree T_2 is the product $D(\rho_1, N(\rho_1))\ W(\rho_1)$.
7. The distance between the weighted tree T_1 and the unweighted tree T_2 is given by the following:

$$\mathcal{D}(T_1, T_2) = \sum_{\rho \in T_1} \Big(D(\rho, N(\rho))\ W(\rho) \Big) \tag{10}$$

Suggest an algorithm to efficiently compute the distance between a weighted tree T_1 and an unweighted tree T_2.

Problem 5: Satisfiability

In the class, we have repeatedly studied the *satisfiability* problem. Given a Boolean formula in the CNF (Conjunctive Normal Form), devise an algorithm for deciding if the formula is satisfiable. You may use the benchmarks available at `http://www.satcompetition.org` for studying the performance of your algorithm. The state-of-the-art can solve problems with 10,000 variables.

You are welcome to explore parallel GPU or CPU based strategies for solving the satisfiability problem.