CS294-3: Algorithms in the real world | Final Exam
Due: 5pm Friday Dec 5, 1997

This is an open book, take home exam. You are free to use any readings, but cannot talk to other people about questions on the exam. I will be in my office much of the time from around 11AM-6PM on Thursday and 12:30-5:00pm on Friday if you have questions. You should read your email regularly to check for any corrections or clarifications I might need to make to the final, and you should drop the final off at my office when you are done.

Problem 1: (Compression: move to front) The move-to-front heuristic is a simple compression heuristic that can take advantage of patterns in data. The idea is to keep the symbols to be compressed ordered. Whenever you read a new symbol to compress, output its position in the ordered symbols and then move it to the front. For example if we start with symbols ordered as $\{A, B, C, D, E\}$, with A at the front, the string $BBCAEB$ would be converted to $\{1, 0, 2, 2, 4, 3\}$. After generating these numbers they can be compressed using Huffman or arithmetic coding (we assume we can see the whole sequence before compressing it). The hope is that there are many small numbers due to "temporal locality" of the symbols so the compression will work well.

In the following questions assume we are using the move-to-front heuristic followed by arithmetic coding.

(A) What do we have to send in order to uncompress the data. (Answer in no more than two sentences.)

(B) If we had a long string with the substring $ABCAB$ repeated over and over, asymptotically how many bits-per-symbol would the move-to-front heuristic use to store the string.

(C) If we had a long string with the substring $ABCDE$ repeated over and over, asymptotically how many bits-per-symbol would the move-to-front heuristic use to store the string.

(D) Would the move-to-front heuristic adapt well to changes in types of text within a single "document" (*e.g.* compressing a tar file containing a mix of English, C and postscript files).

(E) Would coding pairs of symbols (i.e. keep all possible pairs ordered, and code two symbols at a time) help get better compression for long messages? Assume we are compressing the English language.

(F) Joe hacker decides to use the move-to-front heuristic for private-key encryption by using the initial order of the symbols as a key (*i.e* only the sender and receiver know the initial ordering). Would this be easy to crack? If so, give a brief description. Again, assume he is encoding the English language.

Problem 2: (Linear Programming: Minimum-cost network flow) Let $G = (V, E)$ be a directed graph and assume that associated with each vertex $i \in [1, \ldots, n]$ is a quantity b_i representing the net flow out of (positive) or into (negative) a vertex. This, for example, can represent the supply from a distribution site if positive, or the demand from a customer or retailer if negative. For each edge $(i, j) \in E$ we have an associated cost $c_{i,j}$ which represents the per unit cost of transporting material (flow) across that edge. The goal of the minimum-cost network flow problem is to minimize the total cost of the flow across the edges while properly supplying the demand vertices (ones with negative b_i). We assume the total demand equals the total supply ($\sum_i b_i = 0$). The minimum-cost network flow problem can be stated as the following linear program:

$$\begin{array}{lll} \text{minimize} & \sum_{(i,j)\in E} c_{i,j} x_{i,j} & \\ \text{subject to} & \sum_{j|(i,j)\in E} x_{i,j} - \sum_{j|(j,i)\in E} x_{j,i} = b_i, & \text{for all } i \in V \\ & x_{i,j} \geq 0 & \text{for all } (i,j) \in E \end{array}$$

where $x_{i,j}$ represent the flow along edge (i, j).

In the matrix A in the constraint $Ax = b$ formed by this linear program, the rows correspond to vertices and the columns correspond to edges (each column corresponding to an edge has a 1 at the source vertex a -1 at the destination vertex and 0s elsewhere). It turns out, however, that the matrix A does not have full rank (it only has $n - 1$ independent rows) and is therefore not in the correct format for the simplex method. However, an easy fix is to add one additional column to the matrix which is all zeros except for a single 1 on an arbitrary row (vertex). This can be thought of as a root edge that goes from that vertex to nowhere.

For the purpose of this problem we will define a cycle as any chain of edges that comes back on itself, even if the edges are not directed in the same directions (*i.e.* if A points to B and C, and B points to C, then A,B and C form a cycle).

(A) Show that the edges corresponding to a set of basis variables in the simplex method for the matrix A cannot include a cycle. Hint: is the corresponding submatrix invertible?

(B) What does this tell you about the final solution in terms of what subsets of the edges can have nonzero flows?

(C) In terms of the graph, what does one step of the simplex method correspond to. I'm just looking for a short one or two sentence answer, not a full description.

Problem 3: (Integer Programming: Steiner Trees) Formulate the minimum steiner tree problem as an integer program. Assume an undirected graph $G = (V, E)$ with edge weights $w(e)$ and a set of demand points $D \subset V$. The integer program must have at most a polynomial number of equations and variables (in the size of G). There are probably many ways to formulate the problem, but a hint for one way is to consider a set of binary variables $x_{i,j,l}$ which are 1 when there is a path from vertex i to vertex j that traverses at most l edges, and 0 otherwise.

Problem 4: (Delaunay Triangulation: Incremental worst case) In class I mentioned that in the worst case adding a single point to a Delaunay triangulation using the incremental method can modify all the existing triangles, and thus take $\Theta(n)$ time. Show (or describe) an example in which this will happen.

Problem 5: (Nbody Problem: Depth of Barnes-Hut) Consider the following 2-dimensional distribution of points that gets exponentially denser as it approaches a line. All points are in the region $[0,1][0,1]$ (*i.e.* in the square between 0 and 1 in the x and y dimensions). We place one point in the region $[0,1][.5,1]$. We place two points in the region $[0,1][.25,.5]$. We then continue placing 2^k points in the region $[0,1][1/2^{k+1}, 1/2^k]$ until we have placed $n-1$ points. Assuming the points are put down pretty much evenly within each region, how deep would the Barnes-Hut tree be for this distribution as a function of n?

Problem 6: (Indexing: Latent semantic indexing and the SVD) Assume you have n documents and for each we have a vector representing weights on each of m terms (you can assume most weights are zero). This information can be represented as a matrix A with n columns and m rows. Lets say we want to index this data using latent semantic indexing so we run an SVD on the matrix and keep the first k dimensions.

Consider the following example with $n=5, m=10$ and $k=2$ which gives us the $A_2 = U_2\Sigma_2 V_2$ decomposition:

$$A_2 = \begin{bmatrix} .2 & .1 \\ .2 & -.2 \\ .2 & -.4 \\ .4 & .1 \\ .1 & -.3 \\ .2 & -.5 \\ .3 & .2 \\ .1 & -.1 \\ .1 & .2 \\ .4 & -.3 \end{bmatrix} \begin{bmatrix} 2.3 & 0 \\ 0 & 1.5 \end{bmatrix} \begin{bmatrix} .2 & .1 & .1 & .3 & .2 \\ .2 & -.2 & -.5 & .1 & -.3 \end{bmatrix}$$

(A) If someone makes a query using the words 4 and 7 (indexed starting at 1 – i.e. word 4 has column value $(.4, .1)$) and wants the documents with cosine measure within .6 in the reduced 2-d space, which documents does she get?

(B) If I initially based my matrix A on the abstracts of articles and now want to include the titles, which includes some more words, it turns out that I can add words to the existing SVD without modifying the existing entries (this is called folding-in). If a new word "wombat" appears in documents 2 and 5 with weights 1 and in the other documents with weight 0, what is the value of the row I should add (fold-in) to U (i.e the location of "wombat" in the 2-d space).

(C) For a large dataset why might a user query in the reduced dimensional space be much more expensive (runtime) than in the original space, even if I don't include the cost of the transformation from the higher dimension to the lower diminsion. In both cases assume we are using the cosine measure to find the k closest documents.

Problem 7: (Clustering: Using compression to cluster) The object of this problem is to design a divisive (hierarchical top down) monothetic algorithm for clustering based on the minimum description length principal. Assume your data is a set of n objects each with m variables, and for simplicity assume the variables have binary values. The goal is to divide the objects into two sets based on one of the variables (this is what monothetic refers to) and then recursively divide each of the subsets based on one of the remaining variables (the two subcalls could partition on different variables) to build a hierarchical clustering.

The goal is to select a clustering that will allow you to get the best compression of your data, given certain assumptions. This is based on the principal that shorter descriptions are better descriptions.

(A) At each node how would divide the data so that you could compress it the most. To simplify things, in calculating the compression you can assume you are only going to make the one divide. Hint: consider the conditional entropy between variables.

(B) One of the big problems of divisive clustering techniques is to decide when to stop dividing (we typically don't want to divide until there is only one object per group). The method of part (A) gives a natural way to make this decision. Explain how you might decide when to stop dividing (i.e. when you are not going to get any further compression).