Organization of Web Information (CS 728) **Midterm Exam**
Computer Science and Engineering **2010-02-19 Friday**
Indian Institute of Technology Bombay **14:00–16:30 SIC301**

NAME ______________________ ROLL ______________

This exam has 7 printed page/s. Write your roll number on **every sheet, because we may take apart your answer book for correction**. Write your answer clearly within the spaces provided and on any last blank page. **If you need more space than is provided, you probably made a mistake in interpreting the question.** Start with rough work elsewhere, but you need not attach rough work. Use the marks alongside each question for time management. **Illogical or incoherent answers are worse than wrong answers or even *no* answer, and may fetch negative credit.** You may not use any computing or communication device during the exam. You may use textbooks, class notes written by you, approved material downloaded **prior to the exam** from the course Web page, course news group, or the Internet, or notes made available by me for xeroxing. If you use class notes from other student/s, you must obtain them **prior to the exam** and **write down his/her/their name/s and roll number/s** here.

1. We will show that a specific formulation for the multiple string alignment problem that comes up in record extraction from HTML is NP-complete.

1(a) The NP-complete **phylogeny** problem is defined as follows:

- A positive integer D,
- A binary hypercube graph $G = (V, E)$ with D dimensions, i.e., nodes labeled from $\{0,1\}^D$, edges connecting nodes whose labels differ in exactly one bit,
- A subset of nodes $S \subset V$ of size n,
- A positive integer b,

and told to decide if there is a tree embedded in G such that the tree includes all nodes in S and has at most b edges. Then it follows that any instance of the phylogeny problem has a trivial solution tree with at most (____ − ____)____ edges. (Complete with justification.)

		2

Each edge represents flipping one bit in the node label. Choose an arbitrary node in $s \in S$ as center. The shortest path from any other node $t \neq s$ to s has at most D edges. Connect all such t to s via shortest paths. Delete arbitrary edges to break any cycles that may exist. The number of remaining edges in the resulting tree is at most $(n-1)D$.

1(b) The **alignment** problem is defined as follows: We are given a set X of k strings over alphabet Σ. Let alphabet $\Gamma = \Sigma \cup \{\Delta\}$ where Δ is a special insert/delete ("indel") symbol. $M \in \mathbb{R}_+^{|\Gamma|\times|\Gamma|}$ is a symmetric symbol edit cost matrix. An alignment over X is specified as follows. Associate each string with a node in a graph (which may have additional nodes). Every node in the graph is labeled with a distinct ~~string~~ label from Γ^P, where $P \geq \max_{x_i \in X} |x_i|$. The node corresponding to string x_i has a label obtained by inserting $P - |x_i|$ indel characters into x_i. Define $\ell(i)$ to be the label of node i, and $\ell(i)_p$ to be the symbol in the pth position of the string label, $1 \leq p \leq P$. The cost of edge (i, j) is $\sum_p M(\ell(i)_p, \ell(j)_p)$. An alignment is specified by a tree containing all k nodes corresponding to input strings. The cost of an alignment is the total cost of the edges in its corresponding tree. In the optimization version of the alignment problem we seek a tree with the smallest cost. In a decision version we ask if there is an alignment with cost at most ~~M~~ B, where ~~M~~ B is a positive real number. Then any optimal solution to the alignment problem has at most ______ nodes (including the k original nodes) and $P \leq$ ______. (Complete with justification. The answer may involve the longest input string.)

		2

Let $\ell = \max_{x_i \in X} |x_i|$ be the length of the longest input string. We will argue that $2k-1$ nodes and $P \leq k\ell$ are adequate. For the first part, observe that a tree with k leaves has at most $k-1$ internal nodes whose degrees are larger than 2. Chain nodes with degree 2 can be shorted out in our setting, because the edge costs are (at most) additive. The second part is trivial: use node labels with $k\ell$ characters, with one block of ℓ characters reserved for each leaf node. At each leaf node, copy the string to the reserved block, with indels in all other positions. As nodes merge, copy their blocks into the merged node's label. (The reason for wanting to show this is to prove that the alignment is in NP.)

1(c) To show that alignment is NP-hard, we will reduce phylogeny to alignment, by defining $\Sigma = \{0, 1\}$ and

- $X =$ ______
- $k =$ ______
- $B =$ ______
- M as defined by this table:

	0	1	Δ
0	___	___	______
1	___	___	______
Δ	______	______	0

(Hint: some of the blanks should be filled with a quantity that depends on part 1a.)

		4

- $X = \utilde{S}$
- $k = \utilde{n}$
- $B = \utilde{b}$
- M as defined by this table:

	0	1	Δ
0	$\utilde{0}$	$\utilde{1}$	$\utilde{(n-1)D}$
1	$\utilde{1}$	$\utilde{0}$	$\utilde{(n-1)D}$
Δ	$\utilde{(n-1)D}$	$\utilde{(n-1)D}$	0

Note that this alignment instance can be created from the phylogeny instance in polynomial time.

1(d) Given the alignment solution, suppose we write down each $\ell(i)$ as a row in a matrix with $|V|$ rows and P columns. Can there be a column that has both Δ and either a 0 or a 1? Justify or give a counterexample.

		2

Suppose there is a column with at least one instance of $0, 1, \Delta$ in it. Then, from the definition of matrix M above, the cost of the alignment is more than $(n-1)D$. However, we know that there is always a solution with cost at most $(n-1)D$.

1(e) It is obvious that, given a low-cost phylogeny we can get a low-cost alignment. Equipped with the argument in part 1d, show the reverse, i.e., given an alignment with low cost, we can map it to a phylogeny with low cost.

		3

From the previous part we know that no column in the alignment solution can have an indel character *and* a 0 or a 1. This means that in all rows of the alignment matrix, the $P-\ell$ indels are all in the same set of columns. By construction, $\ell = D$ here. So we can safely ignore the columns full of indels henceforth. Now consider edge (u, v) in the alignment tree found by the alignment algorithm. Ignoring the indels in $\ell(u)$ and $\ell(v)$, we can map them back to nodes in the D-dimensional hypercube, and choose any of the shortest paths between them in the hypercube in place of the direct (u, v) edge. Given the total alignment (Hamming) cost is at most B, we will be able to find paths in the hypercube corresponding to all edges in the alignment, such that the total number of edges in the paths is at most B. The paths may intersect leading to cycles, and we may have to delete edges, but the number of edges can only decrease. Summarizing, given an alignment tree with cost B, we can find a phylogeny with cost at most B.

2. Recall our notation when discussing max-margin sequence labeling. Suppose there are N features, M states, and T time steps. For a given sequence of features $x_1 \ldots, x_T$ and labels $y_1, \ldots, y_T$, we juxtaposed two matrices $\phi_t(x, y) \in \mathbb{R}^{M\times N}$ and $\psi_t(x, y) \in \mathbb{R}^{M\times M}$, where

$$\phi_t(x,y)[m,n] = [\![\text{feature } n \text{ is fired in state } m \text{ at position } t]\!]$$
$$\psi_t(x,y)[m,m'] = [\![\text{state transitions from } m \text{ to } m' \text{ before position } t]\!]$$

We then defined $\phi = \sum_t \phi_t$, $\psi = \sum_t \psi_t$, and $\mathbf{f}(x, y) = [\phi(x, y)\ \psi(x, y)]$, reinterpreted as a 1d vector with $MN + M^2$ elements. Recall that the inference procedure found $\arg\max_y \mathbf{w}^\top \mathbf{f}(x, y)$ where $\mathbf{w} \in \mathbb{R}^{MN+M^2}$ is a trained model. Standard Viterbi inference amounted to a shortest path in a graph with M rows and $T+1$ columns. The other piece was the loss function $\Delta(y, y')$, used in the *loss-augmented inference* $\arg\max_y \mathbf{w}^\top \mathbf{f}(x, y) + \Delta(y, y^*)$ during training, where x, y^* and $\mathbf{w}$ are fixed.

2(a) Suppose Δ is the Hamming loss, defined as $\Delta(y, y') = \sum_t [\![y_t \neq y'_t]\!]$. Show how to solve the loss-augmented inference problem using dynamic programming, equivalent to a shortest path in an $M \times (T+1)$-node graph. Give complete update steps for the dynamic program.

		1

Let $m \in [1, M]$ be a state/row and $t, t+1$ be successive timesteps/columns. We will address nodes by the 2d coordinate system (m, t) and define the weight of the edge from (m, t) to $(m', t+1)$ in terms of the observed input sequence x and ideal label sequence y^*. There are three contributions to this edge weight:

- w has a component corresponding to $\psi[m, m']$, accumulate this on to the answer.
- For each feature $n \in [1, N]$, w has a component corresponding to $\phi[m, n]$. If feature n fires in position $t+1$, accumulate this component of w on to the answer.
- If $m' \neq y^*_{t+1}$, then accumulate 1 on to the answer.

2(b) Suppose Δ is the all-or-nothing loss, defined as $\Delta(y, y') = [\![(y_1, \ldots, y_T) \neq (y'_1, \ldots, y'_T)]\!] = 1 - \prod_t [\![y_t = y'_t]\!]$. Show how to solve the loss-augmented inference problem using dynamic programming, with detailed update steps. (Hint: Use a suitable $2M \times (T+1)$-node graph.)

		3

In the previous part, every column t has a "correct" node (y_t, t). Compared to the previous part, the complication is that, once, for some column t, we have passed through a node $(m \neq y^*_t, t)$, the path should not be penalized for passing through an incorrect node in a later column. To remember if we have made a mistake, we make two copies of the $M \times (T+1)$ graph, the upper copy and the lower copy. In the upper copy, trace the single correct path $(y^*_0, 0) \to \cdots \to (y^*_T, T)$. The weight on each of these edges consist of the first two of the three kinds of weights described in the previous part (m' is always y^*_t).

Next, from each correct node (y^*_t, t) in the upper copy, add an edge to all incorrect nodes $(m' \neq y^*_{t+1}, t+1)$ in the lower copy. These edges correspond to potential "first mistakes". The weight of such an edge consists of all the three contributions listed in the previous part, specifically, a contribution of 1 in the third category.

The path pays a loss of 1 when it enters the lower copy; no further loss needs to be paid. In the lower copy, add all admissible state transitions, and assign the edge weights using the first two kinds of contributions only.

2(c) Suppose $\Delta(y, y') = \left[\!\left[\sum_t [\![y_t \neq y'_t]\!] \geq 4\right]\!\right]$, i.e., tolerate errors in up to 3 positions. Specify verbally but precisely how to extend the previous approach; you do not need to write down the update expressions.

		2

Extend the technique in the previous part. Instead of two copies, use four copies. Start in the topmost copy. When the first mistake is made, transition to the second copy, and so on. Do not charge a loss until you make a transition from the third to the fourth copy.

2(d) The loss function is similar to Hamming loss, except that a mistake at a position inside a (parenthesized token segment) is twice as serious (i.e., contributes twice the loss to the overall loss) as a mistake outside parenthesis. (Parenthesis are closed properly and never nested.) Specify verbally but precisely how to work with this loss function; you do not need to write down the update expressions.

		2

This is quite trivial: build the graph exactly as in the Hamming loss case, but select the weight contribution of the third type as 1 or 2 depending on whether position $t+1$ is outside or within parenthesis.

2(e) **(Extra credit)** Suppose the label at each position is from an ordinal set $\{0, 1, \ldots, R-1\}$ and define $\Delta(y, y') = \sum_{t,t'} [\![y_t < y'_t]\!][\![y_{t'} > y'_{t'}]\!]$. Either give polynomial-time algorithms for inference and loss-augmented inference, or prove that such algorithms are unlikely to exist under standard complexity theory assumptions. (Use separate sheets of paper to answer.)

3. Which of the following features defined on the token/s that putatively constitute the mention of an entity require a Semi-CRF model and cannot be represented with a sequential CRF with the basic In/Out state encoding, the Begin-Continue-Other ~~End-Unique~~ state encoding, or simple extensions of these encodings that are still first-order Markov? Give precise justifications in each case, along with any tweaks needed in the state space representation.

3(a) Whether there is a specified delimiter to the immediate left or immediate right of the mention.

		1

No need for semi-CRF. Use Begin-Continue-Other states. The new feature is the OR of these two conjunctive predicates:

- Current state y_t is Begin and x_{t-1} is the specified delimiter
- Current state y_t is Begin/Continue, next state y_{t+1} is Other, and x_{t+1} is the specified delimiter.

3(b) The number of capitalized words in the mention of the entity.

		2

No need for a semi-CRF. We will do this by counting 0/1 features. Define a new feature which is the AND of these predicates:

- Current state y_t is Begin/Continue
- Current token x_t is capitalized.

3(c) Whether the mention is enclosed within a pair of open and close parenthesis, assuming parentheses are well-formed, never nested and never left unclosed. However, note that the parentheses need not be immediately before and after the mention. E.g., "(written by H. G. Wells in 1895)".

		2

No need for a semi-CRF, but we need to doctor the state space a bit to avoid semi-CRFs. Let the original states be Begin (B), Continue (C) and None (N). There is an orthogonal binary state, inside parentheses (I) and outside parentheses (O). The cartesian product has 6 states. Examples of favorable state sequences are

- $(N, O), (N, I), (B, I), (C, I), (N, I), (N, O)$
- $(N, O), (N, I), (B, I), (N, O)$

Examples of unfavorable state sequences are

- $(N, O), (N, I), (B, I), (C, O), (N, O)$
- $(N, O), (B, O), (C, I), (N, I), (N, O)$

Basically we need the pattern $(B, I), (C, I)^*, (N, \cdot)$. So we can use a bit to remember if we have seen (B, I). If this bit is set, and we see (C, O), we reset the bit. If we see $(N, \cdot)$ we reset the bit and fire the feature. If we see (C, I) we keep going. Such a bit can be represented easily by making two copies of the state space as before. (There may be easier state representations.)

3(d) Whether the mention has more than one token.

		2

No need for a semi-CRF. Declare a 0/1 feature which is 1 if y_{t-1} = Begin and y_t = Continue, and 0 otherwise.

4. From a well-formed HTML page we can derive its tag tree, also called the document object model or DOM tree T. Each node $t \in T$ has an element tag, and leaf nodes contain a text segment. We have type labels $1, \ldots, K$ (e.g., corresponding to camera manufacturer, camera model, slowest shutter speed, fastest shutter speed, battery life, weight, and price). In addition there is type label 0 indicating "none of the above". The job is to assign one of these $1 + K$ labels to each node of the DOM tree. To avoid inconsistency, we want *valid* labeling, where, if a node is assigned any label in $\{1, \ldots, K\}$, all its ancestors and descendants must be labeled 0. Let y_t be the label of node $t \in T$ and let $\mathbf{y}$ be the collection of all labels. At each node, we define the feature vector $\mathbf{f}(t, y)$.

4(a) Propose, with justification, four features that you expect to be discriminative across the last five example labels above.

		2

- The state is camera manufacturer or model and there is a dictionary match with the text under the DOM node.
- The state is fastest/slowest shutter speed and the text under the DOM node contains numerals and slash (/).
- The state is weight and the text under the DOM node contains numerals and one of the strings 'gram', 'gm', 'ounce', 'oz'.

- The state is price and the text under the DOM node contains numerals and a currency string like AUD, HKD, Rs, etc.

4(b) The overall feature vector is additive: $\mathbf{f}(T,\mathbf{y}) = \sum_{t\in T}\mathbf{f}(t,y_t)$. As usual, inference means finding $\arg\max_{\mathbf{y}} \mathbf{w}^\top \mathbf{f}(T,\mathbf{y})$. Focus on a specific internal node t_0 with children t_1, t_2 (no generality is lost in assuming exactly two children). For each child t_i ($i=1,2$) and each possible label ~~y_{t_i}~~ k assigned to child t_i, say you have computed the (score of the) best labeling of the subtree rooted at t_i. Call this score table $\alpha(t_i,k)$, where $k\in[0,K]$. What additional bit of information is useful to store at t_i and why?

		1

$\beta(t_i,k)$, set to 1 if some node in the subtree of t_i has been labeled with a non-null label, 0 otherwise.

4(c) Show how to update $\alpha(t_0,k_0)$ and any other information at t_0 based on the score tables $\alpha(t_i,k_i)$ and any other information compiled up to t_1 and t_2. Give precise expressions with case statements as needed. Also show how to trace back the best labeling given the final best score at the root.

		3

$$\alpha(t_0,k_0) = \max_{k_1,k_2}\begin{cases} -\infty, & (k_0\neq 0)\wedge(\beta(t_1,k_1)=1\vee\beta(t_2,k_2)=1)\\ \mathbf{w}^\top\mathbf{f}(t_0,k_0)+\sum_{i=1,2}\alpha(t_i,k_i), & \beta(t_1,k_1)=0\wedge\beta(t_2,k_2)=0\end{cases} \tag{1}$$

Suppose, $\alpha(t_0,k_0)$ reaches the maximum at $k_1=k_!^*$ and $k_2=k_2^*$. Then set

$$\beta(t_0,k_0)=\max\{[\![k_0\neq 0]\!],\beta(t_1,k_1^*),\beta(t_2,k_2^*)\}$$

4(d) Instead of designing the dynamic program on the DOM tree from first principles, you can just adapt Segment-CRF. State informally but precisely how to do this.

		2

In Segment-CRF, given a segment starter token j, we need candidate segment ender tokens $i\geq j$. We usually consider any token (offset) j as a candidate segment starter, and $i = j+1, j+2,\ldots,j+W$ for up to some maximum width W. In the current application, this will not be allowed. To enumerage token offset ranges $[j,i]$ where a segment is allowed to start and finish, we need to consider all internal nodes u in the DOM tree. j (i) has to be the leftmost (rightmost) descendant of u. Given a candidate starting offset j, we need to locate all u such that j is the leftmost descendant of u. Then we propose the rightmost descendants of each u as the segment ender i. Effectively we chalk out a cut frontier across the DOM tree. By specification, any node u on the frontier has all descendants and ancestors at state 0. To account for this, we pretend we have first primed all nodes to state 0, and then compensate the objective $\mathbf{w}^\top(\mathbf{f}(u,k)-\mathbf{f}(u,0))$. Everything else about Segment-CRF works as before.

4(e) Propose and justify a reasonable loss function $\Delta(y,y')$ for this labeling task. (Note: there is no single correct answer, but some are better than others; also, you might want to do something special about label 0. Use the camera example for guidance if needed.)

		2

For the purpose of efficient inference, it would be easiest if loss were also additive: $\Delta(y,y') = \sum_{t\in T}\Delta(y_t,y'_t)$. However, 0/1 loss $\Delta(y_t,y'_t)=[\![y_t\neq y'_t]\!]$ may be too symmetric. Depending on recall/precision requirements, it is better to define a general nonnegative $(K+1)\times(K+1)$

loss matrix, with $\Delta(k,k) = 0\,\forall k$. Additive loss may also be somewhat limiting because we cannot give partial credit for the predicted frontier passing close to the correct frontier, e.g., shifted by one edge to immediate parent or child. (We do not expect a design that is non-additive within exam time.)

4(f) For this choice of Δ show how to extend your inference algorithm to a loss-augmented inference algorithm.

		2

We can support additive Δ with a general nonnegative loss matrix. We will augment $\alpha(t_i, k_i)$ to a new variable $\underline{\alpha}(t_i, k_i)$. To compute this, we add a term to (1):

$$\underline{\alpha}(t_i, k_i) = \alpha(t_i, k_i) + \Delta(k_i, y_i^*).$$

k_1^*, k_2^* will generally change, but $\beta(t, k_t)$ is computed as before.

Total: 40