CME 323: Distributed Algorithms and Optimization

Instructor: Reza Zadeh (rezab@stanford.edu)
TA: Robin Brown (rabrown1@stanford.edu)

HW#2 – Due Thursday May 7 (on Gradescope)

1. **List Prefix Sums** As described in class, List Prefix Sums is the task of determining the sum of all the elements before each element in a list. Let us consider the following simple variation.

   - Select each element from the list randomly and independently with probability $1/\log n$ and add it to a set S. Add the head of the list to this set, and mark all these elements in the list.
   - Start from each element $s \in S$, and in parallel traverse the lists until you find the next element in S (by detecting the mark) or the end of the list. For $s \in S$, call this element found in this way $\texttt{next}(s)$. While traversing, calculate the sum from s to $\texttt{next}(s)$ (inclusive of s but exclusive of $\texttt{next}(s)$), and call this $\texttt{sum}(s)$.
   - Create a list by linking each $s \in S$ to $\texttt{next}(s)$ and with each node having weight $\texttt{sum}(s)$.
   - Compute the List Prefix Sums on this list using pointer jumping. Call the result $\texttt{prefixsum}(s)$.
   - Go back to the original list, and again traverse from each s to $\texttt{next}(s)$ starting with the value $\texttt{prefixsum}(s)$ and adding the value at each node to a running sum and writing this into the node. Now all elements in the list should have the correct prefix sum.

   Analyze the work and depth of this algorithm. These should both be given with high probability bounds.

2. **Random Mate on Graphs** In class we described a random-mate technique for determining graph connectivity. Each node flips a coin, and every edge from a head to a tail will attempt to hook the tail into the head (i.e., relabel the tail with the head pointer). Given a d-regular graph on n vertices, i.e. a graph in which every vertex has degree d, what is the expected number of vertices after one contraction step?

3. **Stochastic Gradient Descent**

   (a) In class we proved that gradient descent on L-smooth functions is guaranteed to decrease the function value at each iteration. Stochastic gradient descent, on the other hand, does not have the same guarantee. Provide an example where stochastic gradient descent does not produce a descent step. Specifically, find a function $f(x) = \sum_{i=1}^{m} f_i(x)$, and an iterate x_0 such that for all step sizes, there exist i such that $f(x_1) > f(x_0)$ (where $x + 1 := x_0 - \alpha \nabla f_i(x)$).

   (b) This exercise will guide you through the convergence proof of SGD. As a reminder, we are proving that if there exists a constant G such that $\mathbb{E}[\|\nabla f_i(x)\|^2] \leq G^2$ and $f(x)$ is μ-strongly convex. Then, with step-sizes $\gamma_k = \frac{1}{\mu k}$, we have

$$\mathbb{E}[\|x_k - x_*\|^2] \leq \frac{\max\{\|x_1 - x_*\|^2, \frac{G^2}{\mu^2}\}}{k}.$$

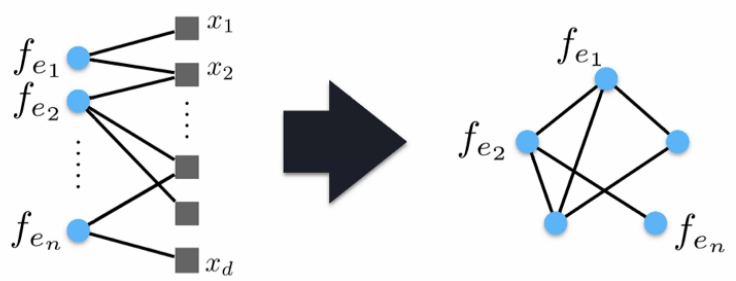

Figure 1: The function-variable and conflict graph for sparse functions.

- Using strong convexity, prove that

$$\langle \nabla f(x_k) - \nabla f(x_*), x_k - x_* \rangle = \langle \nabla f(x_k), x_k - x_* \rangle \geq \mu \left\| x_k - x* \right\|^2$$

- Apply the previous step, to express $\mathbb{E}[\|x_{k+1} - x_*\|^2]$ in terms of $\mathbb{E}[\|x_k - x_*\|^2]$, γ_k, G, and μ.
- Prove the convergence of SGD using induction.

4. **HOGWILD!** This exercise will provide examples applying the main theorem of HOGWILD!. Recall that in HOGWILD!, the objective function we want to minimize is :

$$f(x) = \sum_{e \in E} f_e(x_e)$$

where we define the hyperedge e to be the subset of variables that f_e depends on. Figure 1 depicts such a graph. Then, if we denote the average degree of the conflict graph as $\overline{\Delta}_C$, convergence is still guaranteed if the core delay is less than $\tau \leq \frac{n}{2\overline{\Delta}_C}$ (i.e., no more than τ samples are being processed while a core is processing one).

- **Graph Cuts** In graph cuts problems, we are given a sparse matrix W which indexes similarity between node. We want to match each node to a list of D classes i.e., we want assign a vector $x_i \in \{v \in \mathbb{R}^D | \sum_{j=1}^D v_j = 1, v_j \geq 0\}$ that solve the following optimization problem.

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \sum_{(u,v) \in E} w_{uv} \left\| x_u - x_v \right\|_1 \\ \text{subject to} \quad & x_u \in \{v \in \mathbb{R}^D | \sum_{j=1}^D v_j = 1, v_j \geq 0\}. \end{aligned} \tag{1}$$

Prove that

$$\frac{\overline{\Delta}_C}{n} = \mathcal{O}\left(\text{Avg. deg.}\right)$$

For the next few problems, we expect that you can learn SQL on your own and answer the below questions. Some good guides for learning SQL:

(a) `https://www.w3schools.com/sql/sql_intro.asp`

(b) `https://www.youtube.com/watch?v=9Pzj7Aj25lw`

5. In this set of problems, we review how to *select* data from relational databases.

   (a.) Write a SQL statement to find the total purchase amount of all orders. Sample table: `orders`.

ord_no	purch_amt	ord_date	customer_id	salesman_id
70001	150.5	2012-10-05	3005	5002
70009	270.65	2012-09-10	3001	5005
70002	65.26	2012-10-05	3002	5001
70004	110.5	2012-08-17	3009	5003
70007	948.5	2012-09-10	3005	5002

   (b.) Write a SQL statement which selects the highest grade for each of the cities of the customers. Sample table: `customer`.

customer_id	cust_name	city	grade	salesman_id
3002	Nick Rimando	New York	100	5001
3005	Graham Zusi	California	200	5002
3001	Brad Guzan	London		5005
3004	Fabian Johns	Paris	300	5006
3007	Brad Davis	New York	200	5001

   (c.) Write a SQL statement to find the highest purchase amount on a date "2012-08-17" for each salesman with their ID. Sample table: `orders`, used in (a).

6. In this problem, we review how to *merge* two tables together.

   (a.) Write a SQL statement to know which salesman are working for which customer. Use the sample table `customer`, used in previous problem, and also `salesman`.

salesman_id	name	city	commission
5001	James Hoog	New York	0.15
5002	Nail Knite	Paris	0.13
5005	Pit Alex	London	0.11
5006	Mc Lyon	Paris	0.14
5003	Lauson Hen		0.12

   (b.) Write a query to display all salesmen and customers located in London.

(c.) Write a SQL statement to make a cartesian product between `salesman` and `customer` i.e. each salesman will appear for all customer and vice versa for those salesmen who belongs to a city and the customers who must have a grade.

(d.) Write a SQL statement to make a report with customer name, city, order number, order date, and order amount in ascending order according to the order date to find that either any of the existing customers have placed no order or placed one or more orders. Use `customer` and `orders` tables.

7. In this problem, we consider *aggregation* of data.

   (a.) Write a SQL statement to find the highest purchase amount ordered by the each customer on a particular date with their ID, order date and highest purchase amount. Sample table: `orders`, used in problem 6 part (a).

   (b.) Write a SQL query to display the average price of each company's products, along with their code. Sample table: `item_mast`.

PROD_ID	PROD_NAME	PROD_PRICE	PROD_COMPANY
101	Mother Board	3200	15
102	Key Board	450	16
103	ZIP drive	250	14
104	Speaker	550	16
105	Monitor	5000	11
106	DVD drive	900	12

8. **Joins with multiple keys** The point of this question is to explore how SQL handles cases where a join is performed on tables containing duplicate rows. Consider the following table `item_mast`.

PROD_ID	PRODUCT	PROD_PRICE	PROD_COMPANY
101	Mother Board	3200	1
101	Mother Board	2900	999
103	ZIP drive	250	14
106	DVD drive	900	12

and a corresponding table of customer purchases, `purchases`.

PROD_ID	CUSTOMER	PRODUCT	city
101	James Hoog	Mother Board	New York
101	James Hoog	ZIP drive	Los Angeles
103	Mc Lyon	ZIP drive	Pittsburgh

Notice that in `item_mast`, the same product can appear multiple times (listed under different manufacturers). Also, in database `purchases` the same customer can appear

multiple times. If we `join` carefully using select columns, we can identify observations uniquely in the resulting output table. However, suppose we join the two tables only on `item_mast: product, product price` and `purchases: customer, product`.

Draw a sample table describing what the output looks like, and explain the result.

9. Implement logistic regression using tensorflow. Use the following code to generate train and test data. Note that we have set seed (using "random_state=42"). Use cross-entropy loss and gradient descent optimizer with a learning rate of 0.01. Use batch_size of 100, and run for 500 steps. Report the accuracy on test set.