AdWords Placement Problem via Online Bipartite Graph Matching

Estimated time: 8 hours

Problem: We are given a set of advertisers each of whom has a daily budget B_i. When a user performs a query, an ad request is placed *online* and a group of advertisers can then bid for that advertisement slot. The bid of advertiser i for an ad request q is denoted as b_{iq}. We assume that the bids are small with respect to the daily budgets of the advertisers (i.e., for each i and q, $b_{iq} \ll B_i$). Moreover, each advertisement slot can be allocated to at most one advertiser and the advertiser is charged his bid from his/her budget. The objective is to maximize the amount of money received from the advertisers.

For this project, we make the following simplifying assumptions:

1. For the optimal matching (used for calculating the competitive ratio), we will assume everyone's budget is completely used.
2. The bid values are fixed (unlike in the real world where advertisers normally compete by incrementing their bid by 1 cent).
3. Each ad request has just one advertisement slot to display.

Submission Instructions:

Submit your `adwords.py` or `adwords.R` file on Moodle.

Datasets

You are provided with a small dataset called `bidder_dataset.csv`. This dataset contains information about the advertisers. There are four columns: advertiser ID, query that they bid on, bid value for that query, and their total budget (for all the keywords). The total budget is only listed once at the beginning of each advertiser's list.

In addition, the file `queries.txt` contains the sequence of arrivals of the keywords that the advertisers will bid on. These queries will arrive online and a fresh auctioning would be made for each keyword in this list.

Project Requirements

For this project, your task is to implement the **Greedy**, **MSVV**, and **Balance** algorithms as described below. Your code should also calculate the revenue for the given keywords list (in that order) as well as calculate an estimation of a competitive ratio. The competitive ratio is defined as the minimum of ALG/OPT, where ALT is the mean revenue of your algorithm over all possible input sequences and OPT is the optimal matching. To estimate the value of ALG, simply compute the revenue over 100 random permutations of the input sequence and calculate the mean value.

Project Details:

- Use either python or R for your implementation. Your file should be called either `adwords.py` or `adwords.R`.
- Your code must take one argument as input, which will denote which algorithm to run. The input arguments will be given as: `greedy`, `balance`, or `msvv`.
- Your code must write to the console the calculated revenue using the chosen algorithm on `queries.txt` as given as well as the estimation of the competitive ratio. Simply, print these two numbers on two separate lines.
- NOTE: at some points in the algorithms below, two or more advertisers will tie for an advertisement slot. In this case, choose the advertiser with the smallest ID.

Greedy:

1) For each query q
    a) If all neighbors (bidding advertisers for q) have spent their full budgets
        i) continue
    b) Else, match q to the neighbor with the highest bid.

MSVV:

Let x_u be the fraction of advertiser's budget that has been spent up and $\psi(x_u) = 1 - e^{(x_u - 1)}$.

1) For each query q
    a) If all neighbors have spent their full budgets
        i) continue
    b) Else, match q to the neighbor i that has the largest $b_{iq} * \psi(x_u)$ value.

Balance:

1) For each query q
    a) If all neighbors have spent their full budgets
        i) continue
    b) Else, match q to the neighbor with the highest unspent budget.

Example Output:

Using Python with `random.seed(0)` for shuffling queries, the following was obtained.

command	**revenue**	**competitive ratio**
python adwords.py greedy	16731.40	0.87
python adwords.py mssv	17671.00	0.92
python adwords.py balance	12320.20	0.64