COMPSCI 531D Fall 2021
Project 2: Algorithms for Sponsored Keyword Search
Version D

Due Date: November 14, 2021

1 Introduction

This project asks you to implement, analyze, and propose improvements for algorithms that maximize revenue for *sponsored keyword search*, specifically in the AdWords model. In this model, as users of a search engine platform (e.g. Google or Yahoo!) submit search queries, the server holds an auction for the advertising space on the search results webpage to be returned to the user. The bidders are advertisers which place bids for the ad space on result pages for specific keywords of interest. In addition to their bids, the advertisers also submit a *budget* that they are willing to spend. These auctions happen in fractions of a second.

In the simplest model, which we consider first in this project, all advertisers submit their bids for keywords and their budgets at the start of a fixed-time window, say 24 hours, and the platform uses these given bids and budgets for that period. An advertiser may bid differently for each keyword. The goal is to design an algorithm that maximizes the revenue (i.e. the sum of winning bids), subject to the constraint that all advertisers spend within their budgets.

The project is composed of three components: offline algorithms, online algorithms, and extensions. The first component asks you to implement and analyze specific algorithms for the offline version of the problem, where the queries up for auction are known in advance. The second component asks you to implement and analyze specific algorithms for the online version where the auctions are held immediately after a query arrives, and there is limited knowledge, if any, about the stream of incoming queries. Finally, the third component asks you to propose modifications to these algorithms that improve their performance in reasonably realistic scenarios, e.g., in cases where the bids, budgets, or query sequences are from particular distributions, which may be known or unknown ahead of time.

All implementations will be submitted as source code in **Python 3**[1] and all analyses, figures, and discussions will be included on presentation slides; see Section 6 for more details.

2 Problem Formulation

An instance of the problem consists of n advertisers with budgets $B_1, \dots, B_n$ and a stream of m queries. The queries and m are not known at the beginning; each query is revealed one-at-a-time and m is only known when the stream ends. We say that each t-th query occurs at *time step* t. Each query is

[1] All Project 1's were written in Python, so specifically requiring Python should not be restrictive. Please contact Alex (asteiger@cs.duke.edu) if you would like to use a different programming language.

associated with one keyword j out of the r keywords. In addition to the budgets, each i-th adveriser provides a bid $w_{ij} > 0$ (in U.S. dollars) on each j-th keyword that it is interested in. For simplicity, we set $w_{ij} = 0$ if the advertiser does not bid on the keyword so that w_{ij} is defined for all $i \in [n], j \in [r]$.

For a time step t, let $M_i(t)$ be the total money spent by advertiser i at time t. $M_i(0) = 0$. Set $U_i(t) = B_i - M_i(t)$ to be the *unspent* budget at time t. When a query arrives from the stream at time t with the j-th keyword, either no advertisement is placed on the query results, or one of the advertisers in the subset $\{i \mid U_i(t) \geq w_{ij}\}$ is selected for the advertisement. We set $M_i(t+1) = M_i(t) + w_{ij}$ if advertiser i is chosen at time step t; $M_j(t+1) = M_j(t)$ for all $j \neq i$. Suppose a total of m queries arrive. Then the total *revenue* is $\sum_{i=1}^{n} M_i(m+1)$. The goal is to select bidding advertisers for each query, as they arrive, so that the revenue is maximized.

A graph formulation. This problem can be modeled as a constrained max-weight matching[2] problem, as follows. Let $G = (A \cup Q, E)$ be the bipartite graph where A is the set of advertisers, Q is the set of nodes corresponding to the queries that have arrived from the stream so far, which is initially $\varnothing$, and E contains edge (a, q) if advertiser $a \in A$ bids on the keyword in query q with weight equal to the bid. In this setting, whenever a query arrives from the stream, a node is added to Q along with its corresponding incident edges, and at most one such edge is chosen for the solution (a matching). The goal is to select such edges to maximize the overall weight of the matching while satisfying the constraint that the sum of weights (bids) incident to any vertex in A is at most the budget of the corresponding advertiser.

3 Part A: Offline Algorithms

Before we discuss algorithms for the (online) AdWord problem in the next section, we first consider the *offline* variant where the m queries are known ahead-of-time, along with the bids and budgets of the advertisers. This greatly simplifies the problem since the algorithm must not commit to a bid for a query keyword before knowing those that follow it in the stream. In this setting, we use w_{ij} to denote the bid of the i-th advertiser for the keyword in the j-th query (as opposed to being the bid for the j-th keyword, as defined in the previous section).

3.1 Offline: Greedy

Perhaps the simplest approach is to greedily select bids in non-decreasing order while maintaining feasibility — that is, that all advertisers spends within their budgets and each query is sold to at most one advertiser:

1. Initialize $M_i = 0$ for all $i \in [n]$.
2. For each w_{ij} in non-increasing order:
   (a) Select the i-th advertiser as the winner of the j-th query if (i) the j-th query is not already marked “sold” and (ii) $0 < w_{ij} \leq B_i - M_i$; that is, the advertiser has enough remaining money to pay its bid for the query.

[2] Unlike in other matching problems we have studied this far, note that many edges incident to an (advertiser) vertex may be selected.

   (b) If it was selected, then increase M_i by w_{ij} and mark the query as "sold".

3. Report the selected advertiser (if any) for each query with total revenue $\sum_{i=1}^{n} M_i$.

3.2 Offline: Naïve LP rounding

Another approach is to compute an optimal fractional solution to the LP formulation of the problem, then "round" it into an integral solution which corresponds to a solution to our original problem. (We will have an entire lecture on LP rounding later in the course.) A simple rounding scheme is as follows.

For each $i \in [n]$ and $j \in [m]$, the bid w_{ij} by the i-th advertiser for the j-th query is equal to the advertiser's bid for the keyword in the query; many queries may have the same keyword. The variable x_{ij} represents how much the i-th advertiser is (fractionally) selected to win the auction for the j-th query; in particular, if $x_{ij} = 1$ then the advertiser wins the query outright.

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij}x_{ij} \\ \text{subject to} \quad & \sum_{i=1}^{n} x_{ij} \le 1, && j \in [m] \\ & \sum_{j=1}^{m} w_{ij}x_{ij} \le B_i, && i \in [n] \\ & x_{ij} \ge 0 && i \in [n], j \in [m] \end{aligned}$$

The algorithm is as follows:

1. Initialize $M_i = 0$ for all $i \in [n]$.

2. Compute an optimal fractional solution x^* to the LP.

3. For each x^*_{ij} in non-increasing order:

   (a) Select the i-th advertiser as the winner of the j-th query if (i) the j-th query is not already marked "sold" and (ii) $0 < w_{ij} \le B_i - M_i$; that is, the advertiser has enough remaining money to pay its bid for the query.

   (b) If it was selected, then increase M_i by w_{ij}.

4. Report the selected bids, with total revenue equal to $\sum_{i=1}^{n} M_i$.

3.3 What to do (Code)

1. Implement the greedy algorithm for the offline AdWords problem.

2. Implement the naïve LP rounding algorithm for the offline AdWords problem.

3.4 What to do (Slides)

1. Run each offline algorithm on the four provided datasets (see Section 9). Create a 4×2 table displaying the output revenues for each algorithm and instance pair. Add a third column that shows the value of the optimal (fractional) LP solution.

2. Describe the differences in behavior of the algorithms on these inputs that answer for the differences in revenue. You might construct simpler problem instances to showcase the behavior of the algorithms, either from scratch or using the provided generators of the given datasets, `gen.py`.

3. Propose modifications to these simple algorithms that improve the output revenue. What specific structure, if any, should problem instances have in order for the algorithms to perform better?

4 Part B: Online AdWords

In the online setting, the query stream is not known in advance, and we have to commit to an advertiser (if any) immediately after a query arrives.

4.1 Online: Greedy

The following greedy algorithm, discussed in lecture, is similar to the the greedy algorithm for the offline problem, except that the greedy choice is restricted at each time step t to the bids for the query that just arrived. In particular, the algorithm selects the highest among all bids that are within the advertisers' unspent budgets. The intuition is that, if the budgets were so large that the highest-bidding advertisers would not run out of money before the end of the stream is reached, the revenue accrued would be maximized. The algorithm is as follows:

1. Initialize $M_i = 0$ for all $i \in [n]$.

2. At time step $t \leq m$:

   (a) Let j be the index of the keyword in the query.

   (b) Set $C = \{i \mid B_i - M_i \geq w_{ij} > 0\}$. C是feasible set

   (c) If $|C| > 0$:

      i. Set $i^* = \arg\max_{i \in C} w_{ij}$ then select bid w_{i^*j}. 选最大的

      ii. Increase M_{i^*} by w_{i^*j}.

   (d) Otherwise, there are no feasible bids and the query is unmatched.

3. Report the selected bids, with total revenue equal to $\sum_{i=1}^{n} M_i$.

4.2 Online: Weighted-Greedy

As discussed in lecture, we can improve upon the previous algorithm (in the worst-case) by taking the unspent budget of the advertisers into account. The following algorithm is by Mehta *et al.*. The intuition is to discount the bids from advertisers depending on their remaining budgets so that the algorithm prefers bids from those with more money remaining. Specifically, the advertiser i with the highest *weighted* bid is selected, $\phi_i^{(t)} w_{ij}$, where $\phi_i^{(t)}$ is the discount factor at time t:

$$\phi_i^{(t)} = 1 - e^{(M_i/B_i)-1}.$$

Note that the advertiser with the winning bid still pays its (unweighted) bid, w_{ij}. The complete algorithm is as follows, with the only difference from the previous algorithm being in lines (c.i) and (c.iii):

```
1. Initialize $M_i = 0$ and $\phi_i = 1$ for all $i \in [n]$.
2. At time step $t \le m$:
   (a) Let $j$ be the index of the keyword in the query.
   (b) Set $C = \{i \mid B_i - M_i \ge w_{ij} > 0\}$.
   (c) If $|C| > 0$:
       i. Set $i^* = \arg\max_{i \in C} \phi_i w_{ij}$ then select bid $w_{i^*j}$.
       ii. Increase $M_{i^*}$ by $w_{i^*j}$.
       iii. Set $\phi_{i^*} = 1 - e^{(M_{i^*}/B_{i^*})-1}$.
   (d) Otherwise, there are no feasible bids and the query is unmatched.
3. Report the selected bids, with total revenue equal to $\sum_{i=1}^{n} M_i$.
```

4.3 Online: Learning via Dual LPs

A learning-based approach was proposed by Devanur and Hayes, which is similar to the greedy approach in that that the advertiser selected for a given query is chosen by weighted bids, but the weights are chosen differently. Furthermore, the weights do not change as advertisers' bids are selected!

The algorithm works in two phases: a learning phase that determines the weights, and whose length requires knowing (an estimate of) the number of queries in the stream, m at the start. Additionally, the algorithm has a hyperparameter $\varepsilon \in (0,1)$. Thus, (an estimate of) m and ε are to be specified as input. Note that the algorithm need not know what the queries in the stream are, only how many.

In particular, the learned weights for each advertiser correspond to dual variables of the LP relaxation for the AdWords problem, which you solved for in HW 6. The dual LP is below:

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{i=1}^{n} B_i \alpha_i + \sum_{j=1}^{m} \beta_j \\ \text{subject to} & w_{ij}\alpha_i + \beta_j \ge w_{ij}, i \in [n], j \in [m] \\ & \alpha_i, \beta_j \ge 0 \qquad i \in [n], j \in [m] \end{array}$$

The algorithm is as follows:

```
1. (Addl. input): $m$ and $\varepsilon \in (0,1)$.
2. Initialize $M_i = 0$ for all $i \in [n]$.
3. (Phase 1) For time steps $t \leq \varepsilon m$:
    (a) Let $j$ be the index of the keyword in the query.
    (b) Use the first greedy algorithm to select a bid, then update the $M_i$'s accordingly.
4. Construct an instance $\sigma$ of the offline AdWords problem, restricted to the observed $\lfloor \varepsilon m \rfloor$ queries.
5. Compute an optimal (fractional) solution $(\alpha^*, \beta^*)$ to the dual LP for instance $\sigma$, with the caveat that the dual objective function is modified to have the term $\varepsilon B_i \alpha_i$ in place of $B_i \alpha_i$, for all $i \in [n]$; that is, minimize $\sum_{i=1}^{n} \varepsilon B_i \alpha_i + \sum_{j=1}^{\lfloor \varepsilon m \rfloor} \beta_j$.
6. (Phase 2) For time steps $t$, $\varepsilon m < t \leq m$:
    (a) Let $j$ be the index of the keyword in the query.
    (b) Set $C = \{i \mid B_i - M_i \geq w_{ij} > 0\}$.
    (c) If $|C| > 0$:
        i. Set $i^* = \arg\max_{i \in C} (1 - \alpha_i^*) w_{ij}$ then select bid $w_{i^* j}$.
        ii. Increase $M_{i^*}$ by $w_{i^* j}$.
    (d) Otherwise, there are no feasible bids and the query is unmatched.
7. Report the selected bids, with total revenue equal to $\sum_{i=1}^{n} M_i$.
```

4.4 What to do (Code)

1. Implement the greedy algorithm for the online AdWords problem.
2. Implement the weighted-greedy algorithm for the offline AdWords problem.
3. Implement the learning-based algorithm for the online AdWords problem.

Your implementations will necessarily need to take a query stream (possibly represented by a Python generator, list, etc.), and your implementations **must not** read any t'-th query before handling any t-th query, $t < t'$, even if the representation of the stream you are using allows them to be accessed. (There is no advantage to doing so for the algorithms presented in this section; this will be a bigger concern when you propose improvements to the algorithms in Part C.)

4.5 What to do (Slides)

1. Run each online algorithm on the four provided datasets (see Section 9), using $0.05, 0.1, 0.2$ for the ε parameter of the learning-based algorithm. Create a 4×5 table displaying the output revenues for each algorithm and instance pair.

2. Describe the differences in behavior of the algorithms on these inputs that answer for the differences in revenue. You might construct simpler problem instances to showcase the behavior of the algorithms, either from scratch or using the provided generators of the given datasets, `gen.py`.

3. For each dataset, create 100 new instances by randomly shuffling the query stream, then run each online algorithm on these instances. For the learning-based algorithm, use values $0.05, 0.1, 0.2$ for ε. Create a 4×5 table displaying the average and standard deviations of the output revenues for each algorithm and batch of 100 instances.

4. Compare the results between the revenues from the unshuffled inputs and the shuffled inputs. What algorithms benefit from shuffling? Propose reasons why.

5 Part C: Extensions

Propose **two** possible extensions for the **online** AdWords problem, then implement modifications and perform an experimental analysis to see the impact of your changes on running time, quality of the solution, etc.

Below are some examples that might motivate modifications:

1. The version of the problem we consider can be extended to better model online ad platforms used in practice with two modifications, as follows.

   (a) Instead of only auctioning off one ad "slot" per query, it is more common to display multiple ads in a list on the query result pages. Suppose there are $s > 1$ slots, and the platform has s fixed discount factors, $(d_1 = 1) > d_2 > \ldots d_s > 0$, so that when an advertiser wins the k-th ad slot with bid w_{ij}, it pays a discounted price of $d_k w_{ij}$.

   (b) Instead of advertisers paying when their ad is selected to appear on the query results page (called the *pay-per-impression* model), they pay only if the user actually clicks the ad (called the *pay-per-click* model). Suppose the platform has an estimated probability $p_i \in (0,1)$, called the *clickthrough rate*, that an ad for the i-th keyword will be clicked by a user.

   How can the algorithms be extended to maximize (expected) revenue in the pay-per-click model with multiple ad slots?

2. Consider two advertisers a and b during the weighted greedy algorithm, say with $B_a = B_b$ for simplicity. By design of the weights ϕ_a, ϕ_b, it is possible for b to win a query when bidding less than a, even if a's remaining budget can afford its bid, when $M_a > M_b$. One can see that that this behavior is less desirable near the end of the stream (especially on the last query). How could the weighted greedy algorithm be modified to not only decrease the weights ϕ_i's, but also increase them, in order to maximize revenue? Like the learning-based algorithm, it is reasonable to assume (an estimate of) the length m of the query stream is given as input.

3. Suppose the platform does not have access to the advertiser's budgets at the start, and instead, it is only told when an advertiser's remaining budget is too low to pay for its selected bid. The online algorithms, except for the first greedy algorithm, rely on knowing the budgets. How

could an algorithm be designed to use a "guess" or proxy value for the budget, so that it could be expected to perform better than the first greedy algorithm?

4. Every algorithm presented here, with all other things being equal, will always select an advertiser with the highest bid, even if there are similar (but smaller) bids from other advertisers. For an extreme example, suppose there is an advertiser that bids \$0.01 more than the highest bids from its competitors on every query. On the one hand, preferring this advertiser's bids (slightly) increases revenue. On the other hand, this small difference in bids can create a lop-sided, and arguably unfair, allocation of the ad slots, which could deter advertisers from participating in the platform. How could a fairer algorithm be designed so that similar bids have higher chances to be selected, while also keeping maximizing revenue in mind?

The list above is not intended to be exhaustive — please email Alex and Pankaj if you have other ideas or directions you want to explore! You are welcome to research the problem space provided you clearly cite any references in your slides.

5.1 What to do (Code)

1. To keep your previous code for Parts 1-2 intact, first copy all of your source files into a new directory dedicated for Part 3.
2. Implement your proposed changes and experiments to measure the impact of your changes.

As mentioned in Part 2, your algorithms for the online problem **must not** read any t'-th query before handling any t-th query, $t < t'$, even if the representation of the stream you are using allows them to be accessed. An exception can be made for algorithms which require the *distribution* of queries in the stream, without having any knowledge of which query appears before another in the actual query stream.

5.2 What to do (Slides)

1. Clearly state the aim of each of your two extensions.
2. Propose modifications and state your hypotheses for their impact.
3. Compare and contrast your modified algorithm(s) with the baseline algorithms in Part 2 by conducting experiments. You may use our provided datasets or generate your own. Include your empirical data and analysis in your slides.
4. Describe the strengths and weaknesses of your modified algorithm(s). **Be explicit** about what structure or assumptions should hold for the input, about what additional information is required at the start of your algorithm(s) (e.g. the *distribution* or a shuffled ordering of the queries, but not the actual stream of queries itself), or any other descriptors of the settings where you expect your modified algorithm(s) to perform better than the baseline algorithms. Give the most precise and general classification that you can.
5. Discuss whether your hypotheses held. Why or why not?
6. Describe potential avenues for further work towards your stated aims.

6 Requirements

- ***Groups***: The project is to be done in groups of ***four***, ideally composed of two homework pairs.
- ***Programming languages***: For this project, we require all groups to use Python. Please contact Alex (asteiger@cs.duke.edu) if your group would like to use a different language.
- ***Submission***: Exactly ***ONE*** group member will upload ***two*** files to Sakai:
  1. Your presentation slides in `.ppt` or `.pptx` (PowerPoint) format that includes all experimental analyses, discussions, figures, and “speaker’s notes” per slide. The majority of the text should be in the speaker’s notes. Google Slides, OpenOffice, LibreOffice, and Keynote can export to the required formats.
  2. A single ZIP archive that includes all of your source code and a README file that describes all compilation instructions, execution instructions, and the organization of your code (e.g. what each file contains). If you use any datasets that we do not provide, include links to them if they are accessible online, or include the dataset with your submission if you generated them.

7 Evaluation

This project is worth 15% of your overall course grade. The weight of each Part is as follows:

- 30%: Part 1
- 30%: Part 2
- 30%: Part 3
- 10%: Organization and Presentation

For Parts 1-2, you will be evaluated on the correctness of your implementations of the given procedures, the interpretability of your visualizations/figures, and quality of your experiments and analyses.

For Part 3, you will be evaluated on the clarity of your aims, the soundness of your modifications towards your aims, and the detail and effort of your experimental analyses and conclusions.

For the final category, you will be evaluated on the organization of your ZIP archive and how clearly your experimental data, analyses, and discussions are presented in your slides, including text written in the “speaker’s notes.”

8 Getting Started

Begin by downloading the datasets and the Python file, `gen.py`, used to generate them; see Section 9. Then begin testing your implementations of the offline and online algorithms then measure their performance as required in the updated "What to Do (Slides)" sections of Part A (Step 3.4) and Part B (Step 4.5).

See also Section 11 for changes since the first version of his project handout, and Section 10 about the PuLP package for solving LPs.

9 Datasets

We have provided four datasets for you to experiment with, along with our code, gen.py, used to generate them. These datasets are particularly constructed so that differences in the behavior of the algorithms are showcased.

Each dataset represents a single instance of the AdWords problem. The object stored in a dataset is a tuple of the form (budgets,bids,queries), where budgets is a list of n budgets, bids is a $n \times r$ array where bids[i][j] is the bid of the i-th advertiser for the j-th keyword, and queries is a list of m indices of keywords (which represent the keywords in the stream of m queries). As before, bids of 0 represent the advertiser did not bid. The indices in queries may repeat. Note that the actual keywords are not provided; only their indices are needed by the algorithms.

- ds0.pkl: $n = 2$, $r = 2$, and $m = 200$. One advertiser bids 1 on both keywords, and the other advertiser bids on only one keyword for 0.5. There are 100 queries with each keyword, which (unrealistically) come in an exact, alternating pattern. Generated by ds0_gen(100).
- ds1.pkl: $n = 20$, $r = 400$, and $m = 400$. Each keyword appears exactly once in the stream. Each advertiser has budget 20. For each $i, 0 \leq i < n$, there is an advertiser that bids roughly 1 on the first $20i$ keywords in the stream, and does not bid on the rest. The optimal solution has revenue ≈ 400; that is, every keyword is assigned to a bidding advertiser. Generated by ds1_gen(20,20).
- ds2.pkl: $n = 20$, $r = 400$, and $m = 400$. Each keyword appears exactly once in the stream. Each advertiser has budget 20. There are two types of advertisers:

  1. For each $i \leq n/2$, there is an advertiser that bids roughly 1 on the keywords at time $t, 20i \leq t < 20(i+1)$, and does not bid on the rest.
  2. For each $i > n/2$, there is an advertiser that bids roughly 1 on the keywords at time $t, 20i \leq t < 20(i+1)$, and additionally bids 1 on the first $n/2$ keywords in the stream; no bids are placed on any other keyword.

  The optimal solution has revenue ≈ 400; that is, every keyword is assigned to a bidding advertiser. Generated by ds2_gen(20,20).
- ds3.pkl: $n = 20$, $r = 400$, and $m = 4000$. This dataset is constructed to mimic a realistic scenario, where few advertisers have large budgets and bid on many keywords, and the rest have relatively small budgets and bid on few keywords. The bids and queries are generated by ds3_gen(20,1.3,10), and the budgets were generating using the online greedy algorithm; see the bottom of gen.py.

10 LP Solvers

We will use the PuLP package: link. PuLP is an easy-to-use package to describe LPs and serves as an interface to many LP solvers. Following the link above, PuLP is installed easily using Python's pip package manager. It comes bundled with its own LP solver, CBC, which it uses by default. You may choose to install more sophisticated solvers, such as CPLEX (link) or Gurobi (link), both of which offer free academic licenses.

After installing, review the basic semantics of the PuLP package at the link above, and see the example beer distribution problem here: link. Unfortunately, some of the code snippets do not show correctly on that page: see the entire source code for the example here: link. This example has similarities with the problem we want to solve; in particular, there is a variable for each pair of warehouses and bars, and the AdWords LP has a variable for every pair of advertisers and queries. The example is actually more involved than our problem, but the code to solve it is a good example of PuLP. See the PuLP documentation for more details: link.

11 Clarifications and Changes

- The definition of the set C of advertiser indices in the relevant algorithms has changed from $C = \{i \mid B_i - M_i \geq w_{ij}\}$ to $C = \{i \mid B_i - M_i \geq w_{ij} > 0\}$; that is, we have added the constraint that the advertisers have non-zero bids (since a bid of 0 represents no bid). Without this change, the algorithm cannot differentiate between advertisers with weighted bids of 0 and advertisers that did not bid.

- When computing i^* in the relevant algorithms, multiple advertiers might have the maximum (weighted) bid. How you break ties is up to you. Examples of tie-breaking schemes are random selection among the tied advertisers, or to greedily choose a tied advertiser with the greatest remaining budget.

- The stated values of n, r, m were incorrect for dataset `ds0`, which have been fixed.

- Section 4.3 had a few typos: The dual LP had a $\leq$ instead of $\geq$ in the constraint, and the modified objective stated in step 5 of the algorithm has been corrected.