Chapter 8

Advertising on the Web

One of the big surprises of the 21st century has been the ability of all sorts of interesting Web applications to support themselves through advertising, rather than subscription. While radio and television have managed to use advertising as their primary revenue source, most media – newspapers and magazines, for example – have had to use a hybrid approach, combining revenue from advertising and subscriptions.

By far the most lucrative venue for on-line advertising has been search, and much of the effectiveness of search advertising comes from the "adwords" model of matching search queries to advertisements. We shall therefore devote much of this chapter to algorithms for optimizing the way this assignment is done. The algorithms used are of an unusual type; they are greedy and they are "on-line" in a particular technical sense to be discussed. We shall therefore digress to discuss these two algorithmic issues – greediness and on-line algorithms – in general, before tackling the adwords problem.

A second interesting on-line advertising problem involves selecting items to advertise at an on-line store. This problem involves "collaborative filtering," where we try to find customers with similar behavior in order to suggest they buy things that similar customers have bought. This subject will be treated in Section 9.3.

8.1 Issues in On-Line Advertising

In this section, we summarize the technical problems that are presented by the opportunities for on-line advertising. We begin by surveying the types of ads found on the Web.

8.1.1 Advertising Opportunities

The Web offers many ways for an advertiser to show their ads to potential customers. Here are the principal venues.

1. Some sites, such as eBay, Craig's List or auto trading sites allow advertisers to post their ads directly, either for free, for a fee, or a commission.

2. Display ads are placed on many Web sites. Advertisers pay for the display at a fixed rate per *impression* (one display of the ad with the download of the page by some user). Normally, a second download of the page, even by the same user, will result in the display of a different ad and is a second impression.

3. On-line stores such as Amazon show ads in many contexts. The ads are not paid for by the manufacturers of the product advertised, but are selected by the store to maximize the probability that the customer will be interested in the product. We consider this kind of advertising in Chapter 9.

4. Search ads are placed among the results of a search query. Advertisers bid for the right to have their ad shown in response to certain queries, but they pay only if the ad is clicked on. The particular ads to be shown are selected by a complex process, to be discussed in this chapter, involving the search terms that the advertiser has bid for, the amount of their bid, the observed probability that the ad will be clicked on, and the total budget that the advertiser has offered for the service.

8.1.2 Direct Placement of Ads

When advertisers can place ads directly, such as a free ad on Craig's List or the "buy it now" feature at eBay, there are several problems that the site must deal with. Ads are displayed in response to query terms, e.g., "apartment Palo Alto." The Web site can use an inverted index of words, just as a search engine does (see Section 5.1.1) and return those ads that contain all the words in the query. Alternatively, one can ask the advertiser to specify parameters of the ad, which are stored in a database. For instance, an ad for a used car could specify the manufacturer, model, color, and year from pull-down menus, so only clearly understood terms can be used. Queryers can use the same menus of terms in their queries.

Ranking ads is a bit more problematic, since there is nothing like the links on the Web to tell us which ads are more "important." One strategy used is "most-recent first." That strategy, while equitable, is subject to abuse, where advertisers post small variations of their ads at frequent intervals. The technology for discovering ads that are too similar has already been covered, in Section 3.4.

An alternative approach is to try to measure the attractiveness of an ad. Each time it is displayed, record whether or not the queryer clicked on it. Presumably, attractive ads will be clicked on more frequently than those that are not. However, there are several factors that must be considered in evaluating ads:

1. The position of the ad in a list has great influence on whether or not it is clicked. The first on the list has by far the highest probability, and the probability drops off exponentially as the position increases.

2. The ad may have attractiveness that depends on the query terms. For example, an ad for a used convertible would be more attractive if the search query includes the term "convertible," even though it might be a valid response to queries that look for that make of car, without specifying whether or not a convertible is wanted.

3. All ads deserve the opportunity to be shown until their click probability can be approximated closely. If we start all ads out with a click probability of 0, we shall never show them and thus never learn whether or not they are attractive ads.

8.1.3 Issues for Display Ads

This form of advertising on the Web most resembles advertising in traditional media. An ad for a Chevrolet run in the pages of the *New York Times* is a display ad, and its effectiveness is limited. It may be seen by many people, but most of them are not interested in buying a car, just bought a car, don't drive, or have another good reason to ignore the ad. Yet the cost of printing the ad was still borne by the newspaper and hence by the advertiser. An impression of a similar ad on the Yahoo! home page is going to be relatively ineffective for essentially the same reason. The fee for placing such an ad is typically a fraction of a cent per impression.

The response of traditional media to this lack of focus was to create newspapers or magazines for special interests. If you are a manufacturer of golf clubs, running your ad in *Golf Digest* would give you an order-of-magnitude increase in the probability that the person seeing your ad would be interested in it. This phenomenon explains the existence of many specialized, low-circulation magazines. They are able to charge much more per impression for an ad than is a general-purpose outlet such as a daily newspaper. The same phenomenon appears on the Web. An ad for golf clubs on `sports.yahoo.com/golf` has much more value per impression than does the same ad on the Yahoo! home page or an ad for Chevrolets on the Yahoo! golf page.

However, the Web offers an opportunity to tailor display ads in a way that hardcopy media cannot: it is possible to use information about the user to determine which ad they should be shown, regardless of what page they are looking at. If it is known that Sally likes golf, then it makes sense to show her an ad for golf clubs, regardless of what page she is looking at. We could determine Sally's love for golf in various ways:

1. She may belong to a golf-related group on Facebook.

2. She may mention "golf" frequently in emails posted on her gmail account.

3. She may spend a lot of time on the Yahoo! golf page.

4. She may issue search queries with golf-related terms frequently.

5. She may bookmark the Web sites of one or more golf courses.

Each of these methods, and many others like these, raise enormous privacy issues. It is not the purpose of this book to try to resolve those issues, which in practice probably have no solution that will satisfy all concerns. On the one hand, people like the free services that have recently become advertising-supported, and these services depend on advertising being much more effective than conventional ads. There is a general agreement that, if there must be ads, it is better to see things you might actually use than to have what pages you view cluttered with irrelevancies. On the other hand, there is great potential for misuse if the information leaves the realm of the machines that execute advertising algorithms and get into the hands of real people.

8.2 On-Line Algorithms

Before addressing the question of matching advertisements to search queries, we shall digress slightly by examining the general class to which such algorithms belong. This class is referred to as "on-line," and they generally involve an approach called "greedy." We also give, in the next section, a preliminary example of an on-line greedy algorithm for a simpler problem: maximal matching.

8.2.1 On-Line and Off-Line Algorithms

Typical algorithms work as follows. All the data needed by the algorithm is presented initially. The algorithm can access the data in any order. At the end, the algorithm produces its answer. Such an algorithm is called *off-line*.

However, there are times when we cannot see all the data before our algorithm must make some decisions. Chapter 4 covered stream mining, where we could store only a limited amount of the stream, and had to answer queries about the entire stream when called upon to do so. There is an extreme form of stream processing, where we must respond with an output after each stream element arrives. We thus must decide about each stream element knowing nothing at all of the future. Algorithms of this class are called *on-line* algorithms.[1]

As the case in point, selecting ads to show with search queries would be relatively simple if we could do it off-line. We would see a month's worth of search queries, and look at the bids advertisers made on search terms, as well as their advertising budgets for the month, and we could then assign ads to

[1]Unfortunately, we are faced with another case of dual meanings, like the coincidence involving the term "cluster" that we noted in Section 7.6.6, where we needed to interpret properly phrases such as "algorithms for computing clusters on computer clusters." Here, the term "on-line" refers to the nature of the algorithm, and should not be confused with "on-line" meaning "on the Internet" in phrases such as "on-line algorithms for on-line advertising."

the queries in a way that maximized both the revenue to the search engine and the number of impressions that each advertiser got. The problem with off-line algorithms is that most queryers don't want to wait a month to get their search results.

Thus, we must use an on-line algorithm to assign ads to search queries. That is, when a search query arrives, we must select the ads to show with that query immediately. We can use information about the past, e.g., we do not have to show an ad if the advertiser's budget has already been spent, and we can examine the *click-through rate* (fraction of the time the ad is clicked on when it is displayed) that an ad has obtained so far. However, we cannot use anything about future search queries. For instance, we cannot know whether there will be lots of queries arriving later and using search terms on which this advertiser has made higher bids.

Example 8.1: Let us take a very simple example of why knowing the future could help. A manufacturer A of replica antique furniture has bid 10 cents on the search term "chesterfield".[2] A more conventional manufacturer B has bid 20 cents on both the terms "chesterfield" and "sofa." Both have monthly budgets of \$100, and there are no other bidders on either of these terms. It is the beginning of the month, and a search query "chesterfield" has just arrived. We are allowed to display only one ad with the query.

The obvious thing to do is to display B's ad, because they bid more. However, suppose there will be lots of search queries this month for "sofa," but very few for "chesterfield." Then A will never spend its \$100 budget, while B will spend its full budget even if we give the query to A. Specifically, if there will be at least 500 more queries for either "sofa" or "chesterfield," then there is no harm, and potentially a benefit, in giving the query to A. It will still be possible for B to spend its entire budget, while we are increasing the amount of A's budget that will be spent. Note that this argument makes sense both from the point of view of the search engine, which wants to maximize total revenue, and from the point of view of both A and B, who presumably want to get all the impressions that their budgets allow.

If we could know the future, then we would know how many more "sofa" queries and how many more "chesterfield" queries were going to arrive this month. If that number is below 500, then we want to give the query to B to maximize revenue, but if it is 500 or more, then we want to give it to A. Since we don't know the future, an on-line algorithm cannot always do as well as an off-line algorithm. □

8.2.2 Greedy Algorithms

Many on-line algorithms are of the *greedy algorithm* type. These algorithms make their decision in response to each input element by maximizing some function of the input element and the past.

[2] A chesterfield is a type of sofa. See, for example, `www.chesterfields.info`.

Example 8.2: The obvious greedy algorithm for the situation described in Example 8.1 is to assign a query to the highest bidder who still has budget left. For the data of that example, what will happen is that the first 500 "sofa" or "chesterfield" queries will be assigned to B. At that time, B runs out of budget and is assigned no more queries. After that, the next 1000 "chesterfield" queries are assigned to A, and "sofa" queries get no ad and therefore earn the search engine no money.

The worst thing that can happen is that 500 "chesterfield" queries arrive, followed by 500 "sofa" queries. An off-line algorithm could optimally assign the first 500 to A, earning \$50, and the next 500 to B, earning \$100, or a total of \$150. However, the greedy algorithm will assign the first 500 to B, earning \$100, and then has no ad for the next 500, earning nothing. □

8.2.3 The Competitive Ratio

As we see from Example 8.2, an on-line algorithm need not give as good a result as the best off-line algorithm for the same problem. The most we can expect is that there will be some constant c less than 1, such that on any input, the result of a particular on-line algorithm is at least c times the result of the optimum off-line algorithm. The constant c, if it exists, is called the *competitive ratio* for the on-line algorithm.

Example 8.3: The greedy algorithm, on the particular data of Example 8.2, gives a result that is 2/3 as good as that of the optimum algorithm: \$100 versus \$150. That proves that the competitive ratio is no greater than 2/3. But it could be less. The competitive ratio for an algorithm may depend on what kind of data is allowed to be input to the algorithm. Even if we restrict inputs to the situation described in Example 8.2, but with the bids allowed to vary, then we can show the greedy algorithm has a competitive ratio no greater than 1/2. Just raise the bid by A to ϵ less than 20 cents. As ϵ approaches 0, the greedy algorithm still produces only \$100, but the return from the optimum algorithm approaches \$200. We can show that it is impossible to do worse than half the optimum in this simple case, so the competitive ratio is indeed 1/2. However, we'll leave this sort of proof for later sections. □

8.2.4 Exercises for Section 8.2

! **Exercise 8.2.1:** A popular example of the design of an on-line algorithm to minimize the competitive ratio is the *ski-buying problem*.[3] Suppose you can buy skis for \$100, or you can rent skis for \$10 per day. You decide to take up skiing, but you don't know if you will like it. You may try skiing for any number of days and then give it up. The merit of an algorithm is the cost per day of skis, and we must try to minimize this cost.

[3]Thanks to Anna Karlin for this example.

One on-line algorithm for making the rent/buy decision is "buy skis immediately." If you try skiing once, fall down and give it up, then this on-line algorithm costs you \$100 per day, while the optimum off-line algorithm would have you rent skis for \$10 for the one day you used them. Thus, the competitive ratio of the algorithm "buy skis immediately" is at most 1/10th, and that is in fact the exact competitive ratio, since using the skis one day is the worst possible outcome for this algorithm. On the other hand, the on-line algorithm "always rent skis" has an arbitrarily small competitive ratio. If you turn out to really like skiing and go regularly, then after n days, you will have paid \$10n or \$10/day, while the optimum off-line algorithm would have bought skis at once, and paid only \$100, or \$100/n per day.

Your question: design an on-line algorithm for the ski-buying problem that has the best possible competitive ratio. What is that competitive ratio? *Hint*: Since you could, at any time, have a fall and decide to give up skiing, the only thing the on-line algorithm can use in making its decision is how many times previously you have gone skiing.

8.3 The Matching Problem

We shall now take up a problem that is a simplified version of the problem of matching ads to search queries. This problem, called "maximal matching," is an abstract problem involving *bipartite graphs* (graphs with two sets of nodes – left and right – with all edges connecting a node in the left set to a node in the right set. Figure 8.1 is an example of a bipartite graph. Nodes 1, 2, 3, and 4 form the left set, while nodes a, b, c, and d form the right set.

8.3.1 Matches and Perfect Matches

Suppose we are given a bipartite graph. A *matching* is a subset of the edges such that no node is an end of two or more edges. A matching is said to be *perfect* if every node appears in the matching. Note that a matching can only be perfect if the left and right sets are of the same size. A matching that is as large as any other matching for the graph in question is said to be *maximal*.

Example 8.4: The set of edges $\{(1, a),\ (2, b),\ (3, d)\}$ is a matching for the bipartite graph of Fig. 8.1. Each member of the set is an edge of the bipartite graph, and no node appears more than once. The set of edges

$$\{(1, c),\ (2, b),\ (3, d),\ (4, a)\}$$

is a perfect matching, represented by heavy lines in Fig. 8.2. Every node appears exactly once. It is, in fact, the sole perfect matching for this graph, although some bipartite graphs have more than one perfect matching. The matching of Fig. 8.2 is also maximal, since every perfect matching is maximal. □

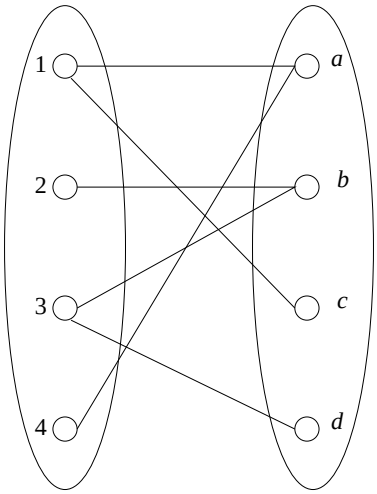

Figure 8.1: A bipartite graph

8.3.2 The Greedy Algorithm for Maximal Matching

Off-line algorithms for finding a maximal matching have been studied for decades, and one can get very close to $O(n^2)$ for an n-node graph. On-line algorithms for the problem have also been studied, and it is this class of algorithms we shall consider here. In particular, the greedy algorithm for maximal matching works as follows. We consider the edges in whatever order they are given. When we consider (x, y), add this edge to the matching if neither x nor y are ends of any edge selected for the matching so far. Otherwise, skip (x, y).

Example 8.5 : Let us consider a greedy match for the graph of Fig. 8.1. Suppose we order the nodes lexicographically, that is, by order of their left node, breaking ties by the right node. Then we consider the edges in the order $(1, a)$, $(1, c)$, $(2, b)$, $(3, b)$, $(3, d)$, $(4, a)$. The first edge, $(1, a)$, surely becomes part of the matching. The second edge, $(1, c)$, cannot be chosen, because node 1 already appears in the matching. The third edge, $(2, b)$, is selected, because neither node 2 nor node b appears in the matching so far. Edge $(3, b)$ is rejected for the match because b is already matched, but then $(3, d)$ is added to the match because neither 3 nor d has been matched so far. Finally, $(4, a)$ is rejected because a appears in the match. Thus, the matching produced by the greedy algorithm for this ordering of the edges is $\{(1, a), (2, b), (3, d)\}$. As we saw, this matching is not maximal. □

Example 8.6 : A greedy match can be even worse than that of Example 8.5. On the graph of Fig. 8.1, any ordering that begins with the two edges $(1, a)$ and $(3, b)$, in either order, will match those two pairs but then will be unable to match nodes 2 or 4. Thus, the size of the resulting match is only 2. □

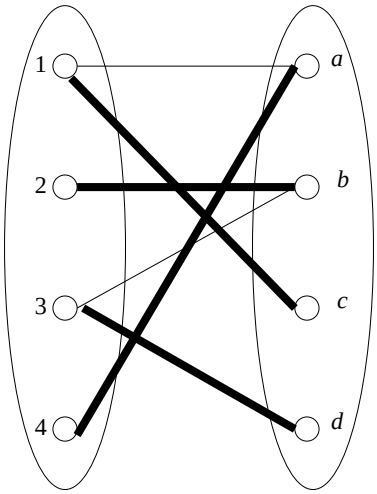

Figure 8.2: The only perfect matching for the graph of Fig. 8.1

8.3.3 Competitive Ratio for Greedy Matching

We can show a competitive ratio of 1/2 for the greedy matching algorithm of Section 8.3.2. First, the ratio cannot be more than 1/2. We already saw that for the graph of Fig. 8.1, there is a perfect matching of size 4. However, if the edges are presented in any of the orders discussed in Example 8.6, the size of the match is only 2, or half the optimum. Since the competitive ratio for an algorithm is the minimum over all possible inputs of the ratio of what that algorithm achieves to the optimum result, we see that 1/2 is an upper bound on the competitive ratio.

Suppose M_o is a maximal matching, and M_g is the matching that the greedy algorithm produces. Let L be the set of left nodes that are matched in M_o but not in M_g. Let R be the set of right nodes that are connected by edges to any node in L. We claim that every node in R is matched in M_g. Suppose not; in particular, suppose node r in R is not matched in M_g. Then the greedy algorithm will eventually consider some edge (ℓ, r), where ℓ is in L. At that time, neither end of this edge is matched, because we have supposed that neither ℓ nor r is ever matched by the greedy algorithm. That observation contradicts the definition of how the greedy algorithm works; that is, the greedy algorithm would indeed match (ℓ, r). We conclude that every node in R is matched in M_g.

Now, we know several things about the sizes of sets and matchings.

1. $|M_o| \leq |M_g| + |L|$, since among the nodes on the left, only nodes in L can be matched in M_o but not M_g.

2. $|L| \leq |R|$, because in M_o, all the nodes in L were matched.

3. $|R| \leq |M_g|$, because every node in R is matched in M_g.

Now, (2) and (3) give us $|L| \leq |M_g|$. That, together with (1), gives us $|M_o| \leq 2|M_g|$, or $|M_g| \geq \frac{1}{2}|M_o|$. The latter inequality says that the competitive ratio is at least 1/2. Since we already observed that the competitive ratio is no more than 1/2, we now conclude the ratio is exactly 1/2.

8.3.4 Exercises for Section 8.3

Exercise 8.3.1: Define the graph G_n to have the $2n$ nodes

$$a_0, a_1, \ldots, a_{n-1}, b_0, b_1, \ldots, b_{n-1}$$

and the following edges. Each node a_i, for $i = 0, 1, \ldots, n-1$, is connected to the nodes b_j and b_k, where

$$j = 2i \mod n \text{ and } k = (2i+1) \mod n$$

For instance, the graph G_4 has the following edges: (a_0, b_0), (a_0, b_1), (a_1, b_2), (a_1, b_3), (a_2, b_0), (a_2, b_1), (a_3, b_2), and (a_3, b_3).

(a) Find a perfect matching for G_4.

(b) Find a perfect matching for G_5.

!! (c) Prove that for every n, G_n has a perfect matching.

! **Exercise 8.3.2:** How many perfect matchings do the graphs G_4 and G_5 of Exercise 8.3.1 have?

! **Exercise 8.3.3:** Whether or not the greedy algorithm gives us a perfect matching for the graph of Fig. 8.1 depends on the order in which we consider the edges. Of the 6! possible orders of the six edges, how many give us a perfect matching? Give a simple test for distinguishing those orders that do give the perfect matching from those that do not.

8.4 The Adwords Problem

We now consider the fundamental problem of search advertising, which we term the "adwords problem," because it was first encountered in the Google Adwords system. We then discuss a greedy algorithm called "Balance" that offers a good competitive ratio. We analyze this algorithm for a simplified case of the adwords problem.

8.4.1 History of Search Advertising

Around the year 2000, a company called Overture (later bought by Yahoo!) introduced a new kind of search. Advertisers bid on *keywords* (words in a search query), and when a user searched for that keyword, the links to all the advertisers who bid on that keyword are displayed in the order highest-bid-first. If the advertiser's link was clicked on, they paid what they had bid.

That sort of search was very useful for the case where the search queryer really was looking for advertisements, but it was rather useless for the queryer who was just looking for information. Recall our discussion in Section 5.1.1 about the point that unless a search engine can provide reliable responses to queries that are for general information, no one will want to use the search engine when they are looking to buy something.

Several years later, Google adapted the idea in a system called *Adwords*. By that time, the reliability of Google was well established, so people were willing to trust the ads they were shown. Google kept the list of responses based on PageRank and other objective criteria separate from the list of ads, so the same system was useful for the queryer who just wanted information as well as the queryer looking to buy something.

The Adwords system went beyond the earlier system in several ways that made the selection of ads more complex.

1. Google would show only a limited number of ads with each query. Thus, while Overture simply ordered all ads for a given keyword, Google had to decide which ads to show, as well as the order in which to show them.

2. Users of the Adwords system specified a budget: the amount they were willing to pay for all clicks on their ads in a month. These constraints make the problem of assigning ads to search queries more complex, as we hinted at in Example 8.1.

3. Google did not simply order ads by the amount of the bid, but by the amount they expected to receive for display of each ad. That is, the click-through rate was observed for each ad, based on the history of displays of that ad. The value of an ad was taken to be the product of the bid and the click-through rate.

8.4.2 Definition of the Adwords Problem

Of course, the decision regarding which ads to show must be made on-line. Thus, we are only going to consider on-line algorithms for solving the *adwords problem*, which is as follows.

- Given:
    1. A set of bids by advertisers for search queries.
    2. A click-through rate for each advertiser-query pair.

3. A budget for each advertiser. We shall assume budgets are for a month, although any unit of time could be used.
4. A limit on the number of ads to be displayed with each search query.

- Respond to each search query with a set of advertisers such that:
  1. The size of the set is no larger than the limit on the number of ads per query.
  2. Each advertiser has bid on the search query.
  3. Each advertiser has enough budget left to pay for the ad if it is clicked upon.

The *revenue* of a selection of ads is the total value of the ads selected, where the *value* of an ad is the product of the bid and the click-through rate for the ad and query. The merit of an on-line algorithm is the total revenue obtained over a month (the time unit over which budgets are assumed to apply). We shall try to measure the competitive ratio for algorithms, that is, the minimum total revenue for that algorithm, on any sequence of search queries, divided by the revenue of the optimum off-line algorithm for the same sequence of search queries.

8.4.3 The Greedy Approach to the Adwords Problem

Since only an on-line algorithm is suitable for the adwords problem, we should first examine the performance of the obvious greedy algorithm. We shall make a number of simplifications to the environment; our purpose is to show eventually that there is a better algorithm than the obvious greedy algorithm. The simplifications:

(a) There is one ad shown for each query.

(b) All advertisers have the same budget.

(c) All click-through rates are the same.

(d) All bids are either 0 or 1. Alternatively, we may assume that the value of each ad (product of bid and click-through rate) is the same.

The greedy algorithm picks, for each search query, any advertiser who has bid 1 for that query. The competitive ratio for this algorithm is 1/2, as the following example shows.

Example 8.7: Suppose there are two advertisers A and B, and only two possible queries, x and y. Advertiser A bids only on x, while B bids on both x and y. The budget for each advertiser is 2. Notice the similarity to the situation in Example 8.1; the only differences are the fact that the bids by each advertiser are the same and the budgets are smaller here.

Adwords Aspects not in Our Model

There are several ways in which the real AdWords system differs from the simplified model of this section.

Matching Bids and Search Queries: In our simplified model, advertisers bid on sets of words, and an advertiser's bid is eligible to be shown for search queries that have exactly the same set of words as the advertiser's bid. In reality, Google, Yahoo!, and Microsoft all offer advertisers a feature known as *broad matching*, where an ad is eligible to be shown for search queries that are inexact matches of the bid keywords. Examples include queries that include a subset or superset of keywords, and also queries that use words with very similar meanings to the words the advertiser bid on. For such broad matches, search engines charge the advertiser based on complicated formulas taking into account how closely related the search query is to the advertiser's bid. These formulas vary across search engines and are not made public.

Charging Advertisers for Clicks: In our simplified model, when a user clicks on an advertiser's ad, the advertiser is charged the amount they bid. This policy is known as a *first-price auction*. In reality, search engines use a more complicated system known as a *second-price auction*, where each advertiser pays approximately the bid of the advertiser who placed immediately behind them in the auction. For example, the first-place advertiser for a search might pay the bid of the advertiser in second place, plus one cent. It has been shown that second-price auctions are less susceptible to being gamed by advertisers than first-price auctions and lead to higher revenues for the search engine.

Let the sequence of queries be $xxyy$. The greedy algorithm is able to allocate the first two x's to B, whereupon there is no one with an unexpended budget to pay for the two y's. The revenue for the greedy algorithm in this case is thus 2. However, the optimum off-line algorithm will allocate the x's to A and the y's to B, achieving a revenue of 4. The competitive ratio for the greedy algorithm is thus no more than 1/2. We can argue that on any sequence of queries the ratio of the revenues for the greedy and optimal algorithms is at least 1/2, using essentially the same idea as in Section 8.3.3. □

8.4.4 The Balance Algorithm

There is a simple improvement to the greedy algorithm that gives a competitive ratio of 3/4 for the simple case of Section 8.4.3. This algorithm, called the *Balance Algorithm*, assigns a query to the advertiser who bids on the query and has the largest remaining budget. Ties may be broken arbitrarily.

Example 8.8: Consider the same situation as in Example 8.7. The Balance Algorithm can assign the first query x to either A or B, because they both bid on x and their remaining budgets are the same. However, the second x must be assigned to the other of A and B, because they then have the larger remaining budget. The first y is assigned to B, since it has budget remaining and is the only bidder on y. The last y cannot be assigned, since B is out of budget, and A did not bid. Thus, the total revenue for the Balance Algorithm on this data is 3. In comparison, the total revenue for the optimum off-line algorithm is 4, since it can assign the x's to A and the y's to B. Our conclusion is that, for the simplified adwords problem of Section 8.4.3, the competitive ratio of the Balance Algorithm is no more than 3/4. We shall see next that with only two advertisers, 3/4 is exactly the competitive ratio, although as the number of advertisers grows, the competitive ratio lowers to 0.63 (actually $1 - 1/e$) but no lower. □

8.4.5 A Lower Bound on Competitive Ratio for Balance

In this section we shall prove that in the simple case of the Balance Algorithm that we are considering, the competitive ratio is 3/4. Given Example 8.8, we have only to prove that the total revenue obtained by the Balance Algorithm is at least 3/4 of the revenue for the optimum off-line algorithm. Thus, consider a situation in which there are two advertisers, A_1 and A_2, each with a budget of B. We shall assume that each query is assigned to an advertiser by the optimum algorithm. If not, we can delete those queries without affecting the revenue of the optimum algorithm and possibly reducing the revenue of Balance. Thus, the lowest possible competitive ratio is achieved when the query sequence consists only of ads assigned by the optimum algorithm.

We shall also assume that both advertisers' budgets are consumed by the optimum algorithm. If not, we can reduce the budgets, and again argue that the revenue of the optimum algorithm is not reduced while that of Balance can only shrink. That change may force us to use different budgets for the two advertisers, but we shall continue to assume the budgets are both B. We leave as an exercise the extension of the proof to the case where the budgets of the two advertisers are different.

Figure 8.3 suggests how the $2B$ queries are assigned to advertisers by the two algorithms. In (a) we see that B queries are assigned to each of A_1 and A_2 by the optimum algorithm. Now, consider how these same queries are assigned by Balance. First, observe that Balance must exhaust the budget of at least one of the advertisers, say A_2. If not, then there would be some query assigned to neither advertiser, even though both had budget. We know at least one of the advertisers bids on each query, because that query is assigned in the optimum algorithm. That situation contradicts how Balance is defined to operate; it always assigns a query if it can.

Thus, we see in Fig. 8.3(b) that A_2 is assigned B queries. These queries could have been assigned to either A_1 or A_2 by the optimum algorithm. We

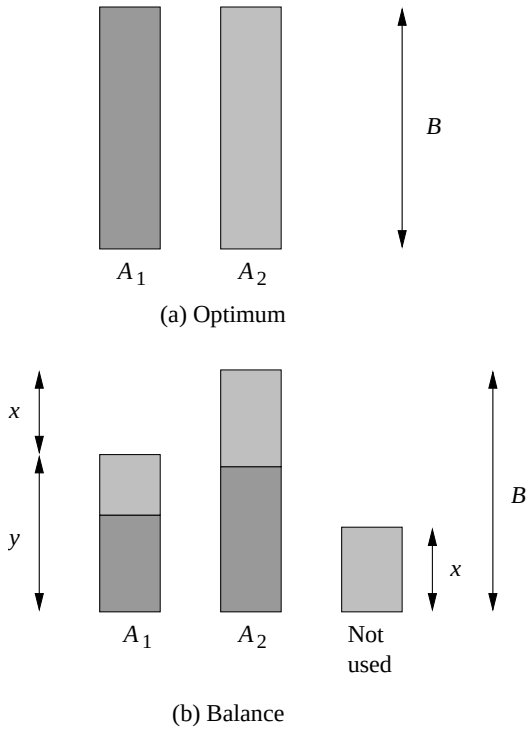

Figure 8.3: Illustration of the assignments of queries to advertisers in the optimum and Balance algorithms

also see in Fig. 8.3(b) that we use y as the number of queries assigned to A_1 and x as $B - y$. It is our goal to show $y \geq x$. That inequality will show the revenue of Balance is at least $3B/2$, or 3/4th the revenue of the optimum algorithm.

We note that x is also the number of unassigned queries for the Balance Algorithm, and that all the unassigned queries must have been assigned to A_2 by the optimum algorithm. The reason is that A_1 never runs out of budget, so any query assigned by the optimum algorithm to A_1 is surely bid on by A_1. Since A_1 always has budget during the running of the Balance Algorithm, that algorithm will surely assign this query, either to A_1 or to A_2.

There are two cases, depending on whether more of the queries that are assigned to A_1 by the optimum algorithm are assigned to A_1 or A_2 by Balance.

1. Suppose at least half of these queries are assigned by Balance to A_1. Then $y \geq B/2$, so surely $y \geq x$.

2. Suppose more than half of these queries are assigned by Balance to A_2. Consider the last of these queries q that is assigned to A_2 by the Balance Algorithm. At that time, A_2 must have had at least as great a budget

available as A_1, or else Balance would have assigned query q to A_1, just as the optimum algorithm did. Since more than half of the B queries that the optimum algorithm assigns to A_1 are assigned to A_2 by Balance, we know that when q was assigned, the remaining budget of A_2 was less than $B/2$. Therefore, at that time, the remaining budget of A_1 was also less than $B/2$. Since budgets only decrease, we know that $x < B/2$. It follows that $y > x$, since $x + y = B$.

We conclude that $y \geq x$ in either case, so the competitive ratio of the Balance Algorithm is 3/4.

8.4.6 The Balance Algorithm with Many Bidders

When there are many advertisers, the competitive ratio for the Balance Algorithm can be under 3/4, but not too far below that fraction. The worst case for Balance is as follows.

1. There are N advertisers, $A_1, A_2, \ldots, A_N$.
2. Each advertiser has a budget $B = N!$.
3. There are N queries $q_1, q_2, \ldots, q_N$.
4. Advertiser A_i bids on queries $q_1, q_2, \ldots, q_i$ and no other queries.
5. The query sequence consists of N rounds. The ith round consists of B occurrences of query q_i and nothing else.

The optimum off-line algorithm assigns the B queries q_i in the ith round to A_i for all i. Thus, all queries are assigned to a bidder, and the total revenue of the optimum algorithm is NB.

However, the Balance Algorithm assigns each of the queries in round 1 to the N advertisers equally, because all bid on q_1, and the Balance Algorithm prefers the bidder with the greatest remaining budget. Thus, each advertiser gets B/N of the queries q_1. Now consider the queries q_2 in round 2. All but A_1 bid on these queries, so they are divided equally among A_2 through A_N, with each of these $N-1$ bidders getting $B/(N-1)$ queries. The pattern, suggested by Fig. 8.4, repeats for each round $i = 3, 4, \ldots$, with A_i through A_N getting $B/(N-i+1)$ queries.

However, eventually, the budgets of the higher-numbered advertisers will be exhausted. That will happen at the lowest round j such that

$$B\Big(\frac{1}{N} + \frac{1}{N-1} + \cdots + \frac{1}{N-j+1}\Big) \geq B$$

that is,

$$\frac{1}{N} + \frac{1}{N-1} + \cdots + \frac{1}{N-j+1} \geq 1$$

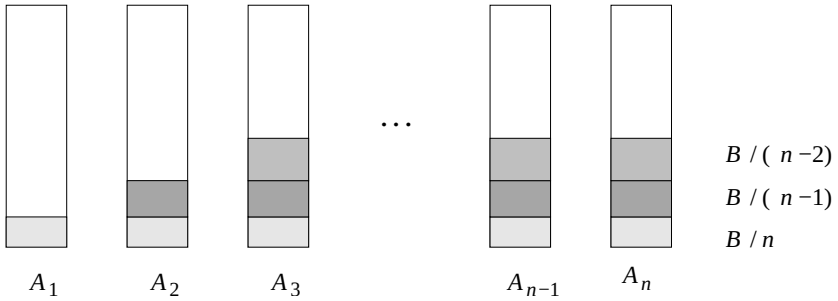

Figure 8.4: Apportioning queries to N advertisers in the worst case

Euler showed that as k gets large, $\sum_{i=1}^{k} 1/i$ approaches $\log_e k$. Using this observation, we can approximate the above sum as $\log_e N - \log_e(N-j)$.

We are thus looking for the j such that $\log_e N - \log_e(N-j) = 1$, approximately. If we replace $\log_e N - \log_e(N-j)$ by the equivalent $\log_e\big(N/(N-j)\big)$ and exponentiate both sides of the equation $\log_e\big(N/(N-j)\big) = 1$, we get $N/(N-j) = e$. Solving this equation for j, we get

$$j = N\Big(1 - \frac{1}{e}\Big)$$

as the approximate value of j for which all advertisers are either out of budget or do not bid on any of the remaining queries. Thus, the approximate revenue obtained by the Balance Algorithm is $BN(1-\frac{1}{e})$, that is, the queries of the first j rounds. Therefore, the competitive ratio is $1-\frac{1}{e}$, or approximately 0.63.

8.4.7 The Generalized Balance Algorithm

The Balance Algorithm works well when all bids are 0 or 1. However, in practice, bids can be arbitrary, and with arbitrary bids and budgets Balance fails to weight the sizes of the bids properly. The following example illustrates the point.

Example 8.9: Suppose there are two advertisers A_1 and A_2, and one query q. The bids on q and budgets are:

Bidder	Bid	Budget
A_1	1	110
A_2	10	100

If there are 10 occurrences of q, the optimum off-line algorithm will assign them all to A_2 and gain revenue 100. However, because A_1's budget is larger, Balance will assign all ten queries to A_1 for a revenue of 10. In fact, one can extend this idea easily to show that for situations like this one, there is no competitive ratio higher than 0 that holds for the Balance Algorithm. □

In order to make Balance work in more general situations, we need to make two modifications. First, we need to bias the choice of ad in favor of higher bids. Second, we need to be less absolute about the remaining budget. Rather, we consider the fraction of the budgets remaining, so we are biased toward using some of each advertiser's budget. The latter change will make the Balance Algorithm more "risk averse"; it will not leave too much of any advertiser's budget unused. It can be shown (see the chapter references) that the following generalization of the Balance Algorithm has a competitive ratio of $1 - 1/e = 0.63$.

- Suppose that a query q arrives, advertiser A_i has bid x_i for this query (note that x_i could be 0). Also, suppose that fraction f_i of the budget of A_i is currently unspent. Let $\Psi_i = x_i(1 - e^{-f_i})$. Then assign q to the advertiser A_i such that Ψ_i is a maximum. Break ties arbitrarily.

Example 8.10: Consider how the generalized Balance Algorithm would work on the data of Example 8.9. For the first occurrence of query q,

$$\Psi_1 = 1 \times (1 - e^{-1})$$

since A_1 has bid 1, and fraction 1 of A_1's budget remains. That is,

$$\Psi_1 = 1 - 1/e = 0.63$$

On the other hand, $\Psi_2 = 10 \times (1 - e^{-1}) = 6.3$. Thus, the first q is awarded to A_2.

The same thing happens for each of the q's. That is, Ψ_1 stays at 0.63, while Ψ_2 decreases. However, it never goes below 0.63. Even for the 10th q, when 90% of A_2's budget has already been used, $\Psi_2 = 10 \times (1 - e^{-1/10})$. Recall (Section 1.3.5) the Taylor expansion for $e^x = 1 + x + x^2/2! + x^3/3! + \cdots$. Thus,

$$e^{-1/10} = 1 - \frac{1}{10} + \frac{1}{200} - \frac{1}{6000} + \cdots$$

or approximately, $e^{-1/10} = 0.905$. Thus, $\Psi_2 = 10 \times 0.095 = 0.95$. □

We leave unproved the assertion that the competitive ratio for this algorithm is $1 - 1/e$. We also leave unproved an additional surprising fact: no on-line algorithm for the adwords problem as described in this section can have a competitive ratio above $1 - 1/e$.

8.4.8 Final Observations About the Adwords Problem

The Balance Algorithm, as described, does not take into account the possibility that the click-through rate differs for different ads. It is simple to multiply the bid by the click-through rate when computing the Ψ_i's, and doing so will maximize the expected revenue. We can even incorporate information about

the click-through rate for each ad on each query for which a nonzero amount has been bid. When faced with the problem of allocating a particular query q, we incorporate a factor that is the click-through rate for that ad on query q, when computing each of the Ψ's.

Another issue we must consider in practice is the historical frequency of queries. If, for example, we know that advertiser A_i has a budget sufficiently small that there are sure to be enough queries coming later in the month to satisfy A_i's demand, then there is no point in boosting Ψ_i if some of A_i's budget has already been expended. That is, maintain $\Psi_i = x_i(1 - e^{-1})$ as long as we can expect that there will be enough queries remaining in the month to give A_i its full budget of ads. This change can cause Balance to perform worse if the sequence of queries is governed by an adversary who can control the sequence of queries. Such an adversary can cause the queries A_i bid on suddenly to disappear. However, search engines get so many queries, and their generation is so random, that it is not necessary in practice to imagine significant deviation from the norm.

8.4.9 Exercises for Section 8.4

Exercise 8.4.1: Using the simplifying assumptions of Example 8.7, suppose that there are three advertisers, A, B, and C. There are three queries, x, y, and z. Each advertiser has a budget of 2. Advertiser A bids only on x; B bids on x and y, while C bids on x, y, and z. Note that on the query sequence $xxyyzz$, the optimum off-line algorithm would yield a revenue of 6, since all queries can be assigned.

! (a) Show that the greedy algorithm will assign at least 4 of these 6 queries.

!! (b) Find another sequence of queries such that the greedy algorithm can assign as few as half the queries that the optimum off-line algorithm assigns on that sequence.

!! **Exercise 8.4.2:** Extend the proof of Section 8.4.5 to the case where the two advertisers have unequal budgets.

! **Exercise 8.4.3:** Show how to modify Example 8.9 by changing the bids and/or budgets to make the competitive ratio come out as close to 0 as you like.

8.5 Adwords Implementation

While we should now have an idea of how ads are selected to go with the answer to a search query, we have not addressed the problem of finding the bids that have been made on a given query. As long as bids are for the exact set of words in a query, the solution is relatively easy. However, there are a number of extensions to the query/bid matching process that are not as simple. We shall explain the details in this section.

8.5.1 Matching Bids and Search Queries

As we have described the adwords problem, and as it normally appears in practice, advertisers bid on sets of words. If a search query occurs having exactly that set of words in some order, then the bid is said to match the query, and it becomes a candidate for selection. We can avoid having to deal with word order by storing all sets of words representing a bid in lexicographic (alphabetic) order. The list of words in sorted order forms the hash-key for the bid, and these bids may be stored in a hash table used as an index, as discussed in Section 1.3.2.

Search queries also have their words sorted prior to lookup. When we hash the sorted list, we find in the hash table all the bids for exactly that set of words. They can be retrieved quickly, since we have only to look at the contents of that bucket.

Moreover, there is a good chance that we can keep the entire hash table in main memory. If there are a million advertisers, each bidding on 100 queries, and the record of the bid requires 100 bytes, then we require ten gigabytes of main memory, which is well within the limits of what is feasible for a single machine. If more space is required, we can split the buckets of the hash table among as many machines as we need. Search queries can be hashed and sent to the appropriate machine.

In practice, search queries may be arriving far too rapidly for a single machine, or group of machines that collaborate on a single query at a time, to handle them all. In that case, the stream of queries is split into as many pieces as necessary, and each piece is handled by one group of machines. In fact, answering the search query, independent of ads, will require a group of machines working in parallel anyway, in order that the entire processing of a query can be done in main memory.

8.5.2 More Complex Matching Problems

However, the potential for matching bids to objects is not limited to the case where the objects are search queries and the match criterion is same-set-of-words. For example, Google also matches adwords bids to emails. There, the match criterion is not based on the equality of sets. Rather, a bid on a set of words S matches an email if all the words in S appear anywhere in the email.

This matching problem is much harder. We can still maintain a hash-table index for the bids, but the number of subsets of words in a hundred-word email is much too large to look up all the sets, or even all the small sets of (say) three or fewer words. There are a number of other potential applications of this sort of matching that, at the time of this writing, are not implemented but could be. They all involve *standing queries* – queries that users post to a site, expecting the site to notify them whenever something matching the query becomes available at the site. For example:

1. Twitter allows one to follow all the "tweets" of a given person. However,

it is feasible to allow users to specify a set of words, such as

```
ipod free music
```

and see all the tweets where all these words appear, not necessarily in order, and not necessarily adjacent.

2. On-line news sites often allow users to select from among certain keywords or phrases, e.g., "healthcare" or "Barack Obama," and receive alerts whenever a new news article contains that word or consecutive sequence of words. This problem is simpler than the email/adwords problem for several reasons. Matching single words or consecutive sequences of words, even in a long article, is not as time-consuming as matching small sets of words. Further, the sets of terms that one can search for is limited, so there aren't too many "bids." Even if many people want alerts about the same term, only one index entry, with the list of all those people associated, is required. However, a more advanced system could allow users to specify alerts for sets of words in a news article, just as the Adwords system allows anyone to bid on a set of words in an email.

8.5.3 A Matching Algorithm for Documents and Bids

We shall offer an algorithm that will match many "bids" against many "documents." As before, a *bid* is a (typically small) set of words. A document is a larger set of words, such as an email, tweet, or news article. We assume there may be hundreds of documents per second arriving, although if there are that many, the document stream may be split among many machines or groups of machines. We assume there are many bids, perhaps on the order of a hundred million or a billion. As always, we want to do as much in main memory as we can.

We shall, as before, represent a bid by its words listed in some order. There are two new elements in the representation. First, we shall include a *status* with each list of words. The status is an integer indicating how many of the first words on the list have been matched by the current document. When a bid is stored in the index, its status is always 0.

Second, while the order of words could be lexicographic, we can lower the amount of work by ordering words rarest-first. However, since the number of different words that can appear in emails is essentially unlimited, it is not feasible to order all words in this manner. As a compromise, we might identify the n most common words on the Web or in a sample of the stream of documents we are processing. Here, n might be a hundred thousand or a million. These n words are sorted by frequency, and they occupy the *end* of the list, with the most frequent words at the very end. All words not among the n most frequent can be assumed equally infrequent and ordered lexicographically. Then, the words of any document can be ordered. If a word does not appear on the list of n frequent words, place it at the front of the order, lexicographically. Those

words in the document that do appear on the list of most frequent words appear after the infrequent words, in the reverse order of frequency (i.e., with the most frequent words of the documents ordered last).

Example 8.11 : Suppose our document is

```
'Twas brillig, and the slithy toves
```

"The" is the most frequent word in English, and "and" is only slightly less frequent. Let us suppose that "twas" makes the list of frequent words, although its frequency is surely lower than that of "the" or "and." The other words do not make the list of frequent words.

Then the end of the list consists of "twas," "and," and "the," in that order, since that is the inverse order of frequency. The other three words are placed at the front of the list in lexicographic order. Thus,

```
brillig slithy toves twas and the
```

is the sequence of words in the document, properly ordered. □

The bids are stored in a hash-table, whose hash key is the first word of the bid, in the order explained above. The record for the bid will also include information about what to do when the bid is matched. The status is 0 and need not be stored explicitly. There is another hash table, whose job is to contain copies of those bids that have been partially matched. These bids have a status that is at least 1, but less than the number of words in the set. If the status is i, then the hash-key for this hash table is the $(i+1)$st word. The arrangement of hash tables is suggested by Fig. 8.5. To process a document, do the following.

1. Sort the words of the document in the order discussed above. Eliminate duplicate words.
2. For each word w, in the sorted order:
   (i) Using w as the hash-key for the table of partially matched bids, find those bids having w as key.
   (ii) For each such bid b, if w is the last word of b, move b to the table of matched bids.
   (iii) If w is not the last word of b, add 1 to b's status, and rehash b using the word whose position is one more than the new status, as the hash-key.
   (iv) Using w as the hash key for the table of all bids, find those bids for which w is their first word in the sorted order.
   (v) For each such bid b, if there is only one word on its list, copy it to the table of matched bids.

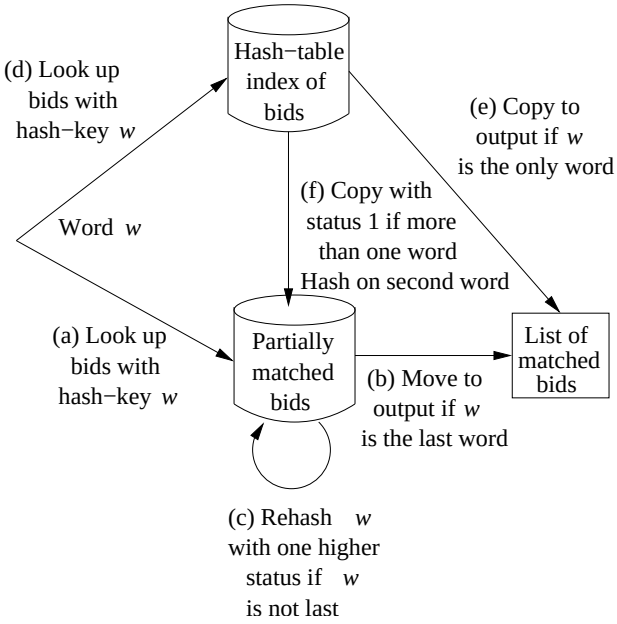

Figure 8.5: Managing large numbers of bids and large numbers of documents

(vi) If b consists of more than one word, add it, with status 1, to the table of partially matched bids, using the second word of b as the hash-key.

3. Produce the list of matched bids as the output.

The benefit of the rarest-first order should now be visible. A bid is only copied to the second hash table if its rarest word appears in the document. In comparison, if lexicographic order was used, more bids would be copied to the second hash table. By minimizing the size of that table, we not only reduce the amount of work in steps 2(i)–2(iii), but we make it more likely that this entire table can be kept in main memory.

8.6 Summary of Chapter 8

Targeted Advertising: The big advantage that Web-based advertising has over advertising in conventional media such as newspapers is that Web advertising can be selected according to the interests of each individual user. This advantage has enabled many Web services to be supported entirely by advertising revenue.

On- and Off-Line Algorithms: Conventional algorithms that are allowed to see all their data before producing an answer are called off-line. An on-

line algorithm is required to make a response to each element in a stream immediately, with knowledge of only the past, not the future elements in the stream.

Greedy Algorithms: Many on-line algorithms are greedy, in the sense that they select their action at every step by minimizing some objective function.

Competitive Ratio: We can measure the quality of an on-line algorithm by minimizing, over all possible inputs, the value of the result of the on-line algorithm compared with the value of the result of the best possible off-line algorithm.

Bipartite Matching: This problem involves two sets of nodes and a set of edges between members of the two sets. The goal is to find a maximal matching – as large a set of edges as possible that includes no node more than once.

On-Line Solution to the Matching Problem: One greedy algorithm for finding a match in a bipartite graph (or any graph, for that matter) is to order the edges in some way, and for each edge in turn, add it to the match if neither of its ends are yet part of an edge previously selected for the match. This algorithm can be proved to have a competitive ratio of 1/2; that is, it never fails to match at least half as many nodes as the best off-line algorithm matches.

Search Ad Management: A search engine receives bids from advertisers on certain search queries. Some ads are displayed with each search query, and the search engine is paid the amount of the bid only if the queryer clicks on the ad. Each advertiser can give a budget, the total amount they are willing to pay for clicks in a month.

The Adwords Problem: The data for the adwords problem is a set of bids by advertisers on certain search queries, together with a total budget for each advertiser and information about the historical click-through rate for each ad for each query. Another part of the data is the stream of search queries received by the search engine. The objective is to select on-line a fixed-size set of ads in response to each query that will maximize the revenue to the search engine.

Simplified Adwords Problem: To see some of the nuances of ad selection, we considered a simplified version in which all bids are either 0 or 1, only one ad is shown with each query, and all advertisers have the same budget. Under this model the obvious greedy algorithm of giving the ad placement to anyone who has bid on the query and has budget remaining can be shown to have a competitive ratio of 1/2.

The Balance Algorithm: This algorithm improves on the simple greedy algorithm. A query's ad is given to the advertiser who has bid on the query and has the largest remaining budget. Ties can be broken arbitrarily.

Competitive Ratio of the Balance Algorithm: For the simplified adwords model, the competitive ratio of the Balance Algorithm is $3/4$ for the case of two advertisers and $1-1/e$, or about 63% for any number of advertisers.

The Balance Algorithm for the Generalized Adwords Problem: When bidders can make differing bids, have different budgets, and have different click-through rates for different queries, the Balance Algorithm awards an ad to the advertiser with the highest value of the function $\Psi = x(1-e^{-f})$. Here, x is the product of the bid and the click-through rate for that advertiser and query, and f is the fraction of the advertiser's budget that remains unspent.

Implementing an Adwords Algorithm: The simplest version of the implementation serves in situations where the bids are on exactly the set of words in the search query. We can represent a query by the list of its words, in sorted order. Bids are stored in a hash table or similar structure, with a hash key equal to the sorted list of words. A search query can then be matched against bids by a straightforward lookup in the table.

Matching Word Sets Against Documents: A harder version of the adwords-implementation problem allows bids, which are still small sets of words as in a search query, to be matched against larger documents, such as emails or tweets. A bid set matches the document if all the words appear in the document, in any order and not necessarily adjacent.

Hash Storage of Word Sets: A useful data structure stores the words of each bid set in the order rarest-first. Documents have their words sorted in the same order. Word sets are stored in a hash table with the first word, in the rarest-first order, as the key.

Processing Documents for Bid Matches: We process the words of the document rarest-first. Word sets whose first word is the current word are copied to a temporary hash table, with the second word as the key. Sets already in the temporary hash table are examined to see if the word that is their key matches the current word, and, if so, they are rehashed using their next word. Sets whose last word is matched are copied to the output.

8.7 References for Chapter 8

[1] is an investigation of the way ad position influences the click-through rate.

The Balance Algorithm was developed in [2] and its application to the adwords problem is from [3].

1. N. Craswell, O. Zoeter, M. Taylor, and W. Ramsey, "An experimental comparison of click-position bias models," *Proc. Intl. Conf. on Web Search and Web Data Mining* pp. 87–94, 2008.

2. B. Kalyanasundaram and K.R. Pruhs, "An optimal deterministic algorithm for b-matching," *Theoretical Computer Science* **233**:1–2, pp. 319–325, 2000.

3. A Mehta, A. Saberi, U. Vazirani, and V. Vazirani, "Adwords and generalized on-line matching," *IEEE Symp. on Foundations of Computer Science*, pp. 264–273, 2005.