Online Bipartite Matching

Seminar Report

Submitted in partial fulfillment of the requirements
for the degree of

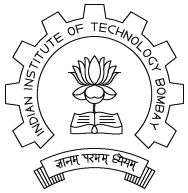
Doctor of Philosophy

by

Jagadish M
Roll No: 09000000

under the guidance of

Prof. Sundar Vishwanathan



Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai

Contents

Abstract

We present a few important results on online bipartite matching problem. The input to the problem is a bipartite graph $G = (U, V, E)$, in which vertices in U arrive in on-line fashion. As each vertex u from U arrives, the edges incident on it are revealed. We need to match u to one of the previously unmatched vertices in V. If no such vertex in V exists, then u remains unmatched. The choice of a match, after it is made, is irrevocable. The objective is to maximize the size of matching resulting after the last vertex arrives. The problem has received considerable attention since it was first introduced by Karp, Vazirani, and Vazirani[3]. Their algorithm, called RANKING, achieves a competitive ratio of $1 - 1/e$. We begin by presenting a simple proof of their result due to Birnbaum[1] and then study a generalized version of the problem in the context of Internet ad-auctions where search queries have to be matched to advertisers in order to maximize the revenue.

Chapter 1

Introduction

1.1 Online algorithms

Generally, in the study of algorithms, the algorithm is assumed to be given the complete input before it can produce the output. However, for many problems arising in practice this is not a reasonable assumption. Some problems may require the algorithm to make a sequence of decisions based on sequentially supplied input data. Hence, an *online* algorithm must generate output without the knowledge of the entire input. Some common examples of online problems are:

Scheduling A set of jobs arriving sequentially must be scheduled on a set of machines, so as to minimize the completion time of the entire set of jobs. Jobs have to be scheduled on machines as they arrive without knowing anything about future jobs.

Routing Routing is the process of choosing paths on which data has to be sent in a communication network. Paths have to be selected as data arrives without the knowledge of future communication patterns.

Paging The problem is to decide which pages to store in the cache memory that are likely to be referenced later, despite not knowing future page requests.

Since online algorithms are forced to make local irrevocable choices, the output generated may not be optimal compared to the *offline* algorithm that has full knowledge about the input. Our interest lies in finding how good the online algorithm performs compared to the offline one. In most cases, the task of the algorithm is to maximize(or minimize) a particular objective function. For example, in job scheduling, we might be interested in minimizing the duration any single machine is used. In paging, we may want to maximize the number of page hits. For a given problem, let $\mathcal{I}$ be the set of all input instances. For an instance $i \in \mathcal{I}$, let $\mathrm{OPT}(i)$ denote the value of the objective function returned by the optimal algorithm. Let $\mathrm{ALG}(i)$ be the value of the objective function returned by the online algorithm ALG. We say ALG is c-competitive if for every instance $i \in I$:

$$\mathrm{ALG}(i) \geq c \cdot \mathrm{OPT}(i)$$

For example, if ALG is 0.5-competitive then on any instance we are assured of obtaining an answer that is at least half as good as the optimal value. Our objective is to design online algorithms with competitive ratios as close to 1 as possible.

A *randomized* algorithm is an algorithm that incorporates certain degree of randomness as part of its strategy in order to achieve a better 'average' case performance. We can think of a randomized algorithm as being a collection of deterministic algorithms(say $\{\mathrm{ALG}_1, \ldots, \mathrm{ALG}_m\}$). When given an input, the

Prob. of picking $\mathbf{ALG_j}$	Competitive ratio
$P(\text{ALG}_1) = 0.6$	$\text{ALG}_1(i) = 0.5$
$P(\text{ALG}_2) = 0.3$	$\text{ALG}_2(i) = 0.6$
$P(\text{ALG}_3) = 0.1$	$\text{ALG}_3(i) = 0.4$

Table 1.1. Expected competitive ratio on instance i is 0.52

randomized algorithm picks one of the deterministic algorithms randomly executes it on the given input. Suppose, we use a randomized algorithm to solve an online problem. The competitive ratio is no more a fixed value. Instead the competitive ratio is a random variable whose value depends on the choice of the deterministic algorithm picked. Therefore, it is reasonable to analyze the performance of a randomized online algorithm by its expected competitive ratio over any instance. The following example illustrates how competitive ratio is calculated: Suppose the randomized algorithm can make three random choices and the relevant values are as shown in Table.1.1 for a particular instance $i \in \mathcal{I}$. The expected competitive ratio on the instance i is

$$E_i[\text{competitive ratio}] = \sum_{j=1}^{3} P(\text{ALG}_j) \times \text{ALG}_j(i)$$

The competitive ratio of the randomized algorithm is the minimum expected value over all input instances:

Competitive ratio$(c) = \min_{i \in \mathcal{I}} E_i[\text{competitive ratio}]$

Usually, the deterministic algorithms in the collection are not explicitly stated, but are implicitly understood by the random choice made. In this chapter, we give a randomized online algorithm that achieves a competitive ratio of 1-1/e for the online bipartite matching problem.

1.2 Problem Definition

We can define the online matching problem as follows:

> Given as input a bipartite graph $G = (U, V, E)$ in which each vertex $u \in U$ arrives in online fashion, devise an algorithm that matches u to one of its previously unmatched neighbours in V[1]. The matching has to be immediate and is irrevocable, once made. The objective is to maximize the size of the resulting matching.

Let $|U| = |V| = n$. Throughout the chapter, we make the assumption that the input graph G has a perfect matching *i.e.* a matching of size n. We denote a perfect matching by a function $m^* : U \to V$. Hence, a c-competitive algorithm must return a matching of size at least cn.

1.3 Deterministic Algorithm

Let us consider the obvious approach first: when u arrives assign it to a some unmatched neighbour. What can we say about the competitive ratio of this algorithm?

Claim 1.3.1. *The above algorithm has a competitive ratio of* $1/2$.

[1] u and v are neighbours if $\{u, v\}$ is an edge in the G

Proof. If a vertex u_1 is not present in the resulting matching $\mathcal{M}$, then it does not have an unmatched neighbour, if not, we would have matched u_1 to that neighbour. Hence, the resulting matching must be maximal. For every edge $\{u, m^*(u)\}$, either vertex u or $m^*(u)$ is present in $\mathcal{M}$. So, at least $n/2$ vertices are matched and hence the algorithm has a competitive ratio of $1/2$. □

We can prove that any deterministic algorithm cannot do better than the obvious algorithm. An adversary can limit the size of matching to $n/2$ in the following way: Let the first $n/2$ vertices that arrive have edges to all the vertices in V. Clearly, the adversary can determine the vertices in V that will be matched. Let the next $n/2$ vertices that arrive contain edges only to those vertices in V which are already matched. The input graph has a perfect matching but the second half of the vertices are not matched; hence our analysis is tight for the deterministic case.

1.4 Randomized Algorithm

A natural question to ask is if we can improve the competitive ratio by making use of randomization. Recall that we calculate the performance of a randomized algorithm in the following manner: for each possible input, calculate the expectation of the answer and take the worst expected value among all the inputs. In our case, an input instance would be specified by a graph G along with an arrival order π. So the input space would contain all possible (G, π) pairs.

The most natural way to introduce randomness would be to match u to one of the unmatched neighbours picked randomly. This algorithm performs better than the deterministic algorithm; however the improvement is not substantial.

Claim 1.4.1. *A randomized algorithm that picks an unmatched neighbour uniformly and randomly has a competitive ratio of at most* $n/2 + O(\log n)$.

Proof. To see why this is the case, consider the following input. Let $\{u_1, \ldots, u_n\} \in U$ and $\{v_1, \ldots, v_n\} \in V$. There is an edge between u_i and v_i for all i. Every vertex in the first half of U $\{u_1, \ldots, u_{n/2}\}$ is connected to every vertex in the second half of V $\{v_{n/2}, \ldots, v_n\}$ as shown below:

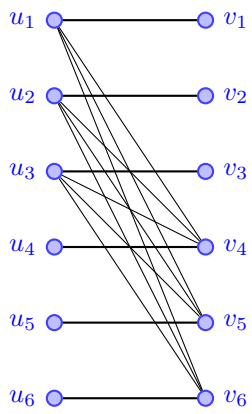


Figure 1.1. Tight instance for the naïve randomized algorithm

The vertices arrive in the order of their indices. Intuitively, the algorithm fails to perform well on this input since it matches too many us from the first half to the vs of the second half.

- The first $n/2$ vertices from U are definitely in the matching since all of them get at least one unmatched neighbour when they arrive.

- Each u_i from the second half of U can be matched to v_i, if v_i is not already matched. What is the probability that this happens?

Let us find the expected number of vertices that are matched in the first half of V. When u_1 arrives, it can pick either v_1 or $\{v_{n/2}, \ldots, v_n\}$. So the probability of v_1 getting matched is $\frac{1}{n/2+1}$. Similarly, when u_2 arrives, the probability that v_2 gets matched is:

$$P(v_2 \text{ gets picked}) = P(v_1 \text{ was picked})\frac{1}{n/2+1} + P(v_1 \text{ was not picked})\frac{1}{n/2}$$
$$P(v_2 \text{ gets picked}) < \frac{1}{n/2}$$

Similarly, probability of v_3 getting matched is less than $1/(n/2-1)$ and in general a vertex v_i in the first half of V has less than $1/(n/2-i+2)$ probability of being in the matching.

Let E_v be the expected number of vertices from $\{v_1, \ldots, v_{n/2}\}$ in the matching.

$$E_v < \frac{1}{n/2+1} + \frac{1}{n/2} + \cdots + \frac{1}{2}$$
$$E_v < O(\log n)$$

Hence, less than $O(\log n)$ unmatched neighbours are expected to be available to the second half of U, which proves our claim. □

However, a slightly different randomized algorithm, named RANKING, due to Karp[**?**] performs much better. The algorithm runs in two steps. In the first step, it chooses a random ranking on vertices in V. In the second step, it matches each vertex upon arrival to an unmatched neighbour deterministically. The algorithm is as follows:

Algorithm: RANKING

Initialization: Pick a random permutation(ranking) σ of the vertices in V

Online Matching:

For each $u \in U$ that arrives:

Let $N(u)$ be the set of unmatched neighbours of u

If $N(u) \neq \emptyset$, match u to the vertex $v \in N(u)$ such that $\sigma(v)$ is minimum.

The RANKING algorithm gives a competitive ratio of $1 - 1/e \approx 0.63$. Before we formally prove this, we shall try to intuitively understand its behaviour. Since the online matching phase of the algorithm is no different than the deterministic algorithm, we are assured that the resulting matching is maximal and therefore 0.5-competitive. Randomization helps in getting a additional 0.13 factor improvement.

Let x_t be the probability over σ that the vertex of V with rank t is in the matching. In other words, for an instance (G, π), how likely is the vertex with rank t in the resulting matching? The algorithm matches u to the unmatched neighbour of smallest rank so the vertex with rank 1 is always present in the matching. Hence, x_1 equals 1. What about x_2? Let $r_i \in V$ be the vertex with rank i. The only competitor to r_2 is r_1, so we are only interested in vertices in U that are connected to both r_1 and r_2. Suppose a vertex u arrives which has an unmatched vertex r_2 as its neighbour. Consider the following cases:

1. Vertex r_1 is unmatched, so u gets matched to r_1
2. Vertex r_1 is matched, so u gets matched to r_2.
3. Vertex r_2 has more than one neighbour, so r_2 definitely gets matched to one of them.

Hence, only in the first case is r_2 unmatched. From the above we can conclude the following. The vertex r_2 is not matched only if all these conditions are met:

1. Vertex r_2 has degree 1.
2. There exists a single vertex $u^1 \in U$ that is connected to both r_1 and r_2.
3. Vertex u^1 is the first vertex to have r_1 as its neighbour.

An instance (G, π) where r_2 is not matched must look like the graph shown in Fig. 1.2. The dashed edges may or may not be present. Since $|V| = n$ there are n vertices that can be picked as r_2. Out of n choices only in one case is the vertex with rank 2 not matched(as shown in the figure). Hence the probability of a r_2 being not matched is less than $1/n$.

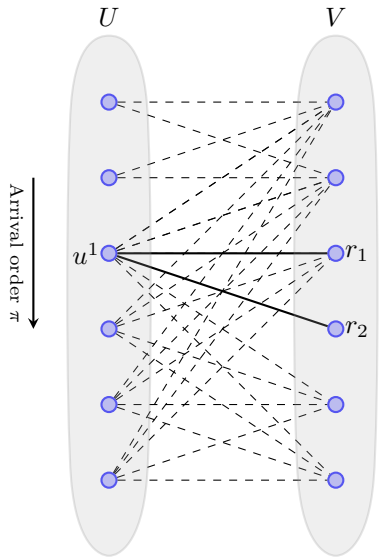


Figure 1.2. Only possible situation where a vertex with rank 2 is not matched

We have,

$$x_1 = 1$$
$$x_2 = 1 - \frac{1}{n} x_1$$

In general, if x_t is the probability over σ that the vertex of V of rank t is matched, then the following inequality holds:

$$x_t \geq 1 - \frac{1}{n} \sum_{1 \leq s \leq t} x_s$$

Or the probability that a vertex or rank t is not matched is:

$$1 - x_t \leq \frac{1}{n} \sum_{1 \leq s \leq t} x_s \tag{1.1}$$

The right hand side of Eq.1.1 is the expected number of vertices matched with rank less than t. The inequality can be interpreted as: "If a vertex of rank t is unlikely to be matched, then there are good number of vertices from higher ranks in the matching". Before we prove Eq.1.1 let us see how this inequality allows us to get a better competitive ratio.

Claim 1.4.2. *If $1 - x_t \leq (1/n)\sum_{1\leq s\leq t} x_s$ holds, then the competitive ratio of* RANKING *is $1 - 1/e$.*

Proof. The expected size of matching is $\sum_{1\leq s\leq n} x_s$. The competitive ratio is the minimum value of this expression. Let $S_t = \sum_{1\leq s\leq t} x_s$. Therefore,

$$\begin{aligned} 1 - (S_t - S_{t-1}) &\leq \frac{1}{n} S_t \\ S_t \left(1 + \frac{1}{n}\right) &\geq 1 + S_{t-1} \end{aligned}$$

Minimum value occurs when all the inequalities are equalities.

$$\begin{aligned} S_n \left(1 + \frac{1}{n}\right) &= 1 + S_{n-1} \\ S_n \left(1 + \frac{1}{n}\right)^2 &= \left(1 + \frac{1}{n}\right) + S_{n-1}\left(1 + \frac{1}{n}\right) \\ S_n \left(1 + \frac{1}{n}\right)^n &= \left(1 + \frac{1}{n}\right)^{n-1} + \left(1 + \frac{1}{n}\right)^{n-2} + \cdots + \left(1 + \frac{1}{n}\right) \\ S_n &\geq \left(1 - \frac{1}{n+1}\right) + \left(1 - \frac{1}{n+1}\right)^2 + \cdots + \left(1 - \frac{1}{n+1}\right)^n \\ S_n &\geq n\left(1 - \left(1 - \frac{1}{n+1}\right)^n\right) \\ S_n &= n\left(1 - \frac{1}{e}\right) \text{ as } n \to \infty \end{aligned}$$

□

We are left to prove that Eq.1.1 holds. Let RANKING(σ) denote the matching constructed on the instance (G, π). Note that once (G, π) and the ranking order σ are specified, the matching is completed determined.

Claim 1.4.3. *Pick a vertex $u \in U$. Let $v = m^*(u)$ and rank of v be t. If v is not matched by* RANKING(σ)*, then u is matched by* RANKING(σ) *to a vertex v' whose rank $\sigma(v')$ is less than t.*

Proof. If v is not matched by RANKING(σ), then v is was one of the unmatched neighbours of u at the time of us arrival. Since u was not matched to v, we can conclude that u had another unmatched neighbour v' with rank less than t. □

Based on the above observation, we give an intuitive(but incorrect!) reasoning for proof of Eq.1.1. Let $v \in V$ be the vertex with rank t. Let u be the vertex such that $u = m^*(v)$. Let $R_{t-1} \subset U$ denote the set of vertices that are matched to vertices with rank less than t. As earlier, let x_t denote the probability that v is matched.

$$\begin{aligned} P(v \text{ is not matched}) &= 1 - x_t \\ v \text{ is not matched} &\Rightarrow u \in R_{t-1} \quad \text{(by Claim 1.4.3)} \\ \therefore P(v \text{ is not matched}) &\leq P(u \in R_{t-1}) \end{aligned} \tag{1.2}$$

We are choosing u such that $u = m^*(v)$. Therefore, u and R_{t-1} are not independent. If u and R_{t-1} were independent, then we could write the probability that a random u belongs to R_{t-1} as:

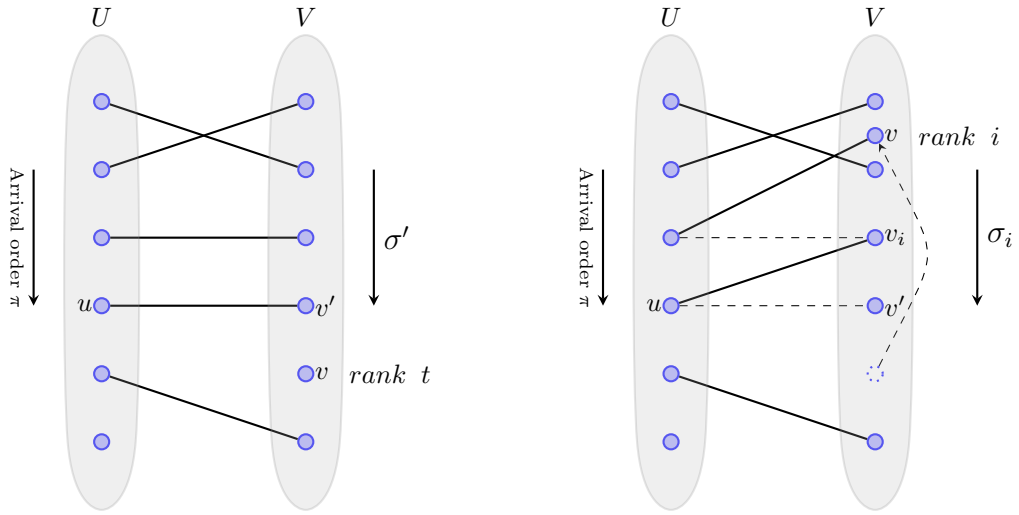


Figure 1.3. RANKING(σ') is shown on the left. Note that $u = m^*(v)$, but u is not connected to v, therefore u must be connected to v' whose rank is smaller than t. On the right is the matching RANKING(σ_i) obtained when v is moved to rank i. Since vertex v's rank is lesser now it became the prefered choice for third node in U, subsequently its neighbour v_i became free. The relative ranking of all the other nodes remain the same when a node is shifted; u got matched to v_i whose rank is less than v'.

$$P(u \in R_{t-1}) = \frac{1}{n}(\text{Expected size of } R_{t-1})$$
$$P(u \in R_{t-1}) = \frac{1}{n} \sum_{1 \le s \le t-1} x_s \tag{1.3}$$

From Eq.1.2 and Eq.1.3, we can prove that Eq.1.1 holds:

$$P(v \text{ is not matched}) \le P(u \in R_{t-1})$$
$$1 - x_t \le \frac{1}{n} \sum_{1 \le s \le t-1} x_s$$

Unfortunately, u and R_{t-1} are not independent and the proof does not go through. To fix that, we take a different, non-intuitive, route to defining probabilities in Eq.1.2.

Claim 1.4.4. *Let $u \in U$ and $v = m^*(u)$. Let σ' be a permutation and let σ_i be the permutation obtained from σ' by removing v and placing it back so that its rank is i. If v is not matched by* RANKING*(σ'), then, for every i, u is matched by* RANKING*(σ_i) to a vertex v_i whose rank $\sigma_i(v_i)$ is at most $t = \sigma'(v)$.*

Proof. Let $m =$RANKING(σ') and $m_i =$RANKING(σ_i). If m and m_i are identical then the claim follows from the earlier Claim.1.4.3. We have two cases:

- If vertex v is moved to rank i greater than t then m is identical to m_i. This is obvious because v was not a the minimum unmatched neighbour for any of the vertices $u \in U$, increasing the rank only lowers the chance of getting matched.
- Suppose vertex v is moved to rank i less than t. In this case, it is possible that v becomes the minimum ranked unmatched neighbour for some vertex in U.(See Fig.1.3). We can easily see that the vertices in U get matched to neighbours of lesser or equal ranks in m_i than they were matched to in m. Hence, if u was matched to v' in m, then m_i matches u to a vertex v_i whose rank is at most that of v'. Hence, $\sigma_i(v_i) \le t$.

□

Proof of Eq.1.1. Based on the above observation, we can now give a correct argument of the earlier proof. Suppose we are given a random permutation σ.

- Pick a vertex v uniformly and randomly from σ. Remove it and replace it such that its rank becomes t. Call this permutation σ'. If $1 - x_t$ is the probability over σ that a vertex is not matched, then the probability that v is not matched in σ' is $1 - x_t$.
- Consider the matching RANKING(σ'). Let u be such that $v = m^*(u)$. If u is not matched by RANKING(σ'), then we can apply Claim.1.4.4 by considering $\sigma_i = \sigma$(assume v had a rank of i in σ) and say that if v is not matched by RANKING(σ'), then u is matched by RANKING(σ') to a vertex $\hat{v}$ such that $\sigma(\hat{v}) \leq t$, or equivalently, $u \in R_t$. Since u is independent of σ it is independent of R_t. So, conditional on σ, the event that probability that $u \in R_t$ is $|R_t|/n$.

The inequality $1 - x_t \leq \frac{1}{n} \sum_{1 \leq s \leq t} x_s$ now holds and hence the proof. □

Chapter 2

Generalized Matching

A natural extension of the online bipartite matching considered in the last chapter is to ask what happens if multiple nodes are allowed to be matched to a single node. For example, can we get a similar performance if each vertex from V is allowed to match to atmost b vertices from U? More specifically we define the problem as follows:

> Given as input a bipartite graph $G = (U, V, E)$ in which each vertex $u \in U$ arrives in online fashion, devise an algorithm that matches u to one of its neighbours v_i in V such that the number of vertices matched to v_i is at most b. The matching, as before, has to be immediate and is irrevocable. The objective is to maximize the number of matched vertices in U.

This problem, called b-Matching, was studied by Kalyanasundaram and Pruhs[2], who gave a deterministic algorithm that achieves a competitive ratio of $1 - \frac{1}{(1+1/b)^b}$ and futher proved a matching lower bound for any deterministic algorithm. Their algorithm, called BALANCE, is quite simple:

BALANCE
For each $u \in U$ that arrives:
Let $N(u)$ be the set of neighbours of u.
Let $c(v')$ denote the number of vertices in U matched to $v' \in V$ so far.
If $N(u) \neq \emptyset$, match u to the vertex $v \in N(u)$ with minimum $c(v)$.

In this chapter, we study the competitive analysis of BALANCE in a more general framework of Adwords problem introduced by Mehta *et al.*[4].

2.1 Adwords Problem

Most Internet search companies derive their revenue from advertisments(Ads). When a user searchs for a query, ads are shown alongside search results and the advertisers are charged each time their ad is shown to(or clicked by) the user. An advertiser has a fixed daily budget and is interested in a small set of queries which can be bought by a bidding process. The goal for the search engine is to show those ads which maximizes the revenue at the end of the day.

The adwords problem in defined as follows:

> There are N bidders, each with a specified daily budget b_i. Q is a set of query words. Each bidder i specifies a bid c_{iq} for query word $q \in Q$. A sequence $q_1, q_2, \ldots, q_M$ of query words $q_j \in Q$ arrive online during the day, and each query q_j must be assigned to some bidder i, (for a revenue of c_{iq_j}). The objective is to maximize the total revenue at the end of the day while respecting the daily budgets of the bidders.

The classic online bipartite matching is the case when all the bidders have a unit budgets and unit bids. The b-Matching problem corresponds to the case when each bidder has a budget of b units and the bids have unit values. We know that any deterministic algorithm cannot acheive a competitive ratio better than $1/2$ for the classic online bipartite matching problem. Here, we make the assumption that each bid is small compared to the corresponding budget, *i.e.* $\max_j c_{ij}$ is small compared to b_i, for all i. Typically, individual bids for a query are small compared to the bidder's budget, hence this is a reasonable assumption. Under this assumption, we derive a $(1 - 1/e)$-competitive deterministic algorithm.

To simplify the proof, we make the following assumptions(which will be removed later):

- The budgets of all bidders are equal to one unit.
- The best offline algorithm exhausts the budget of each bidder. Along with the previous assumption, this means the optimal revenue that can be obtained is N.

Greedy Algorithm

Let us first consider a greedy algorithm that allots each query to the highest bidder. It is easy to see that this algorithm cannot acheive a competitive ratio of better than $1/2$. Here is a tight example: There are two bidders A and B, both with unit budgets, and two query words q_1 and q_2. The two bidders bid $c - \epsilon$ and c, respectively on query word q_1, and they bid 0 and c on query word q_2. The query sequence consists of $1/c$ occurrences of q_1 followed by $1/c$ occurrences of q_2. Greedy algorithm gives all q_1 queries to bidder 2, thereby exhausting his budget. When query words q_2 arrives, bidder 1 is not interested in this query word, and hence get no revenue from bidder 1. If all q_1 type queries were alloted to A and q_2 queries to B, the revenue obtained would be nearly 2 units.

Instead of the picking bidders greedily, we allot queries by taking into consideration the unspent budget of each bidder.

2.1.1 Discretization

To make the analysis simpler, we discretize the budgets as follows: we pick a large integer k, and discretize the budget of each bidder into k equal slabs numbered 1 through k. Each bidder spends money in slab j before moving to slab $j + 1$.

Let slab(i) denote the currently active slab for bidder i, at any time during the run of the algorithm.

Let $\psi(k) : [1 \dots k] \rightarrow \mathbb{R}^+$ be the following (monotonically decreasing) function:

$$\psi(i) = 1 - e^{(1-i/k)}$$

We will discuss the derivation of this trade-off function later.

2.1.2 Discrete Version of the Algorithm

> When a new query arrives, let the bid of bidder i be $c(i)$. Allocate the query to the bidder i who maximizes $c(i) \times \psi_k(\text{slab}(i))$.

Correspondence to b-Matching

If all the bidders have equal budget and the bids are equal, this problem is the same as b-Matching problem. Without loss of generality assume, that each bidder has a budget of 1 unit and each bid is of $1/b$ unit. The correspondence is as follows:

U:Queries that arrive online

V:Bidders. Each bidder has a budget of 1 unit, so each $v \in V$ can connect to b vertices in U.

E:An edge between u and v means the bidder v has a bid of $1/b$ unit for query u.

The algorithm described above works in the same way as the BALANCE does. We analyze the performance of the BALANCE in this case.

2.2 Competitive Analysis of BALANCE

Assume that each bidder has a budget of 1 unit and that in the optimal solution all the bidders exhaust their budgets *i.e.* total revenue that can be obtained is N. (We will remove this assumption later). BALANCE tries to allocate the queries as evenly as possible. Suppose we run BALANCE and look at the allocation done at the end of the algorithm. We assign a *type* to each bidder according to the fraction of budget he has spent. A bidder of type j is one who has spent an amount between $((j-1)/k, j/k]$. For example, for $k = 3$, bidders who have spent fractions 1/6, 1/2, 5/6 of their budget would belong to type 1,2, and 3, respectively. A bidder who has spent no amount also belongs to type 1. At the end of the algorithm, let x_i denote the number of bidders of type i(See Fig.2.1). Let us calculate the revenue BAL obtained at the end of algorithm.

If the bidder of type i has spent less than i/k, let us round up the value to i/k. So we overestimate the revenue from each bidder by at most $1/k$, and consequently the total revenue by N/k. Also assume that each query is paid for exclusively from one slab. The is a reasonable assumption since the bids are small compared to budgets. If we assume bids are smaller than $1/k^2$, then the error in revenue resulting from this assumption is atmost N/k. Assuming this correction, the revenue obtained from x_1 bidders is x_1/k, from x_2 bidders is $2x_2/k$, and so on.

Therefore,

$$\text{BAL} = \sum_{i=1}^{k} \frac{i}{k} x_i$$

We know $\sum_{i=1}^{k} x_i = N$ so

$$\text{BAL} = \sum_{i=1}^{k-1} \frac{i}{k} x_i + \left(N - \sum_{i=1}^{k-1} x_i \right)$$

and we have overestimated the revenue by at most N/k, so

$$\text{BAL} \geq \sum_{i=1}^{k-1} \frac{i}{k} x_i + \left(N - \sum_{i=1}^{k-1} x_i \right) - \frac{N}{k}$$

For small values of j, bidders of type j contribute very little revenue and our analysis upper bounds the number of users of these types. In order to do this, we need one key observation on comparing the allocation of the optimal algorithm to BALANCE. Suppose, we have four bidders b_1, b_2, b_3 and b_4 and k is fixed to 2 and at the end of the algorithm the allocation is as shown in Fig.2.1. Suppose the optimal algorithm assigns a query q to a bidder in type 1 (say b_1) but our algorithm allocates it to some bidder from type 2(say b_2). What can we say about the query during the run of BALANCE? Note that when query q arrived b_1 was a contender for query q, but since the query was alloted to b_2 it must have been the case that b_2 had lesser fraction of his budget spent than b_1. Since, b_1 is of type 1 and spends less than 1/2 of his budget, query q must have been paid from slab 1.

Claim 2.2.1. *If the optimal algorithm assigns query q to a bidder B of type $j \leq k-1$, then* BALANCE *pays for q from some slab i such that $i \leq j$.*

Proof. This follows from the way BALANCE allocates queries. Bidder B belongs to type j and spends at most $j/k \leq 1$ fraction of his budget at the end of BALANCE. When q arrives, B is available to BALANCE for allocating q, hence q must be assigned to some bidder who has spent at most j/k fraction of his budget. □

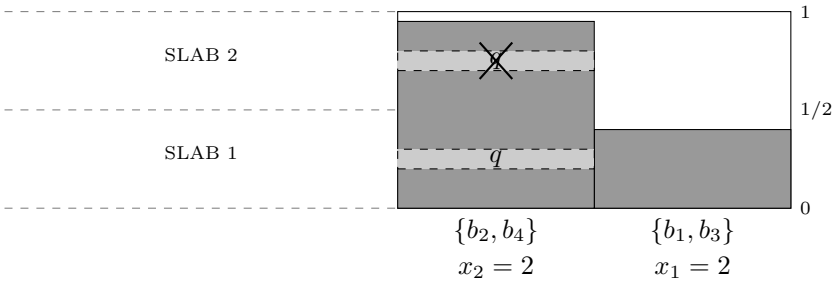


Figure 2.1. If the optimal algorithm gives q to bidder b_1 or b_3, then q must have been paid for from SLAB 1(not from SLAB 2) in the run of BALANCE

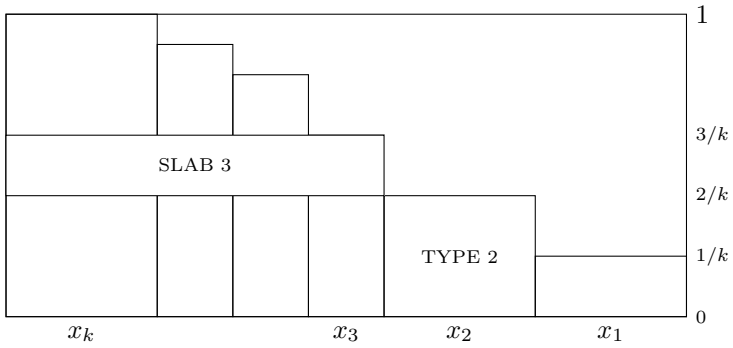


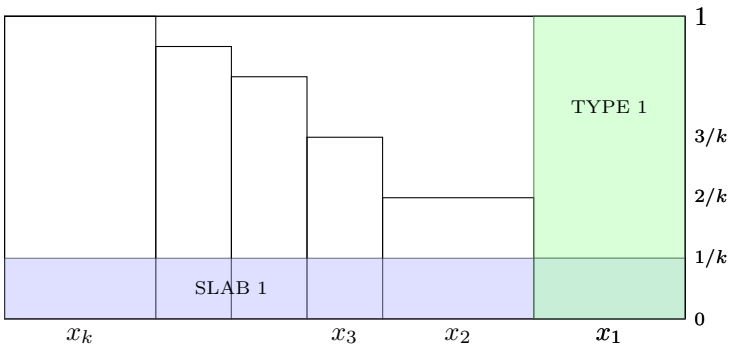
Figure 2.2. Number of bidders of type 1 is x_1, number of bidders of type 2 is x_2 and so on. All bidders have budget of 1 unit. Bidders from type x_2 are those who have spent between $(1/k, 2/k]$ fraction of their budget at the end of BALANCE

Let β_i denote the revenue obtained from slab i during the run of the BALANCE. So, $\beta_1 = N/k$, $\beta_2 = N/k - x_1/k$, and in general

$$\beta_i = \frac{N}{k} - \frac{(x_1 + \ldots + x_{i-1})}{k}$$

For Claim. 2.2.1 we can deduce the relation between x_j and β_j. (See Fig.2.3).

$$\sum_{j=1}^{i} x_j \leq \sum_{j=1}^{i} \beta_j$$



If a query is given to a bidder of type 1 by the optimal algorithm, it is present in slab 1 after the run of BALANCE

Figure 2.3. Relation between x_1 and β_1. In general, we can say $\sum_{j=1}^{i} x_j \leq \sum_{j=1}^{i} \beta_j$.

Claim 2.2.2.

$$\sum_{j=1}^{i}\left(1+\frac{i-j}{k}\right)x_j \le \frac{i}{k}N \qquad\qquad \forall i,\ 1 \le i \le k-1$$

Proof.

$$\begin{aligned}
\beta_1 &= N/k \\
\beta_2 &= N/k - (x_1)/k \\
\beta_3 &= N/k - (x_1+x_2)/k \\
&\vdots \\
\beta_i &= N/k - (x_1+x_2+\cdots+x_{i-1})/k \\
\sum_{j=1}^{i}\beta_j &= \frac{i}{k}N - \sum_{j=1}^{i}\left(\frac{i-j}{k}\right)x_j
\end{aligned}$$

From Claim 2.2.1 we have,

$$\begin{aligned}
\sum_{j=1}^{i} x_j &\le \sum_{j=1}^{i}\beta_j \\
\sum_{j=1}^{i} x_j &\le \frac{i}{k}N - \sum_{j=1}^{i}\left(\frac{i-j}{k}\right)x_j \\
\sum_{j=1}^{i}\left(1+\frac{i-j}{k}\right)x_j &\le \frac{i}{k}N
\end{aligned}$$

□

The revenue of the algorithm is

$$\begin{aligned}
\text{BAL} &\ge \sum_{i=1}^{k-1}\frac{i}{k}x_i + \left(N - \sum_{i=1}^{k-1}x_i\right) - \frac{N}{k} \\
&= N - \sum_{i=1}^{k-1}\frac{k-i}{k}x_i - \frac{N}{k}
\end{aligned}$$

To find the lower bound on the performance of Balance we want to find the minimum value BAL can take over all feasible x_is. We have the following LP L:

$$\begin{aligned}
\text{maximize} \quad & \Phi = \sum_{i=1}^{k-1}\frac{k-i}{k}x_i \\
\text{subject to} \quad & \forall i: \ \sum_{j=1}^{i}\left(1+\frac{i-j}{k}\right)x_j \le \frac{i}{k}N \\
& \forall i: \ x_i \ge 0
\end{aligned}$$

Let us look at the constraints closely.

$$x_1 \leq \frac{1}{k}N$$
$$(1+\frac{1}{k})x_1 + x_2 \leq \frac{2N}{k}$$
$$\vdots$$

Setting inequalities to equalities, we get,

$$x_1 = \frac{1}{k}N$$
$$x_2 = \frac{2N}{k} - \left(1+\frac{1}{k}\right)\frac{N}{k} = \frac{N}{k}\left(1-\frac{1}{k}\right)$$
$$\vdots$$
$$x_i = \frac{N}{k}\left(1-\frac{1}{k}\right)^{i-1}$$

We get a feasible set $x_i^* \geq 0$ which is also the optimal solution since x_is cannot be increased further.

$$x_i^* = \frac{N}{k}\left(1-\frac{1}{k}\right)^{i-1}$$

Hence, the value of the optimal solution is:

$$\Phi = \sum_{i=1}^{k-1}\frac{k-i}{k}x_i^*$$
$$= \sum_{i=1}^{k-1}\left(\frac{k-i}{k}\right)\frac{N}{k}\left(1-\frac{1}{k}\right)^{i-1} \qquad = N\left(1-\frac{1}{k}\right)^{k} = \frac{N}{e} \text{as } k \text{ tends to } \infty$$

Since the revenue obtained(size of matching) is BAL and BAL $\geq N - \Phi - \frac{N}{k}$. We know that BAL tends to $N\left(1-\frac{1}{e}\right)$ as we make the discretization finer. □

2.3 Multiple bid values

Now we consider the more realistic case of all the bids not being equal. We cannot simply allot the query to the bidder with maximum unspent budget since his bid might be very low. We can neither allot the query to the highest bidder since we run into the problem of being greedy and exhausting the bidder's budget. Hence, we seek to obtain a *trade-off* function that takes into account both the bid amount and the bidder's fraction of spent budget to make the decision. We discuss how to derive the optimal trade-off function using trade-off revealing family of LPs. Claim 2.2.1 that allowed us to write inequality constraints in the LP of BALANCE no more holds. Despite this, the formulation of BALANCE is quite helpful but the approach taken is quite non-intuitive.

Let us recall the LP, L, we considered:

$$\begin{aligned}
\text{maximize} \quad & \Phi = \sum_{i=1}^{k-1} \frac{k-i}{k} x_i \\
\text{subject to} \quad & \forall i: \ \sum_{j=1}^{i} \left(1 + \frac{i-j}{k}\right) x_j \leq \frac{i}{k} N \\
& \forall i: \ x_i \geq 0
\end{aligned}$$

The dual can be written as:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{k-1} \frac{i}{k} N y_i \\
\text{subject to} \quad & \forall i: \ \sum_{j=1}^{k-1} \left(1 + \frac{j-i}{k}\right) y_j \geq \frac{k-i}{k} \\
& \forall i: \ y_i \geq 0
\end{aligned}$$

Define $\mathbf{A}, \mathbf{b}, \mathbf{c}$ so that LP, L, can be written in standard form as:

$$\max \ \mathbf{c} \cdot \mathbf{x} \ \ s.t. \ \ \mathbf{A}\mathbf{x} \leq \mathbf{b} \ \ \mathbf{x} \geq \mathbf{0}$$

and the dual D as:

$$\min \ \mathbf{b} \cdot \mathbf{y} \ \ s.t. \ \ \mathbf{A}^\top \mathbf{y} \geq \mathbf{c} \ \ \mathbf{y} \geq \mathbf{0}$$

Just as we set all the inequalities to equalities and get $\mathbf{x}^*$. Similarly, we can set the inequalities in the dual to equalities to get a feasible solution $\mathbf{y}^*$ as:

$$y_i^* = \frac{1}{k}\left(1 - \frac{1}{k}\right)^{k-i-1}$$

The optimal objective function value is:

$$\begin{aligned}
\Phi_d &= \mathbf{b}.\mathbf{y}^* \\
&= \sum_{i=1}^{k-1} \left(\frac{k-i}{k}\right) \frac{N}{k} \left(1 - \frac{1}{k}\right)^{i-1} \\
&= N\left(1 - \frac{1}{k}\right)^{k} = \mathbf{c} \cdot \mathbf{x}^* = \mathbf{\Phi}
\end{aligned}$$

Since, $\mathbf{c} \cdot \mathbf{x}^* = \mathbf{b} \cdot \mathbf{y}^*$, $\mathbf{y}^*$ is the optimal solution to the dual LP, D.

2.3.1 Ensemble of LPs

For a given instance of the problem π and a particular choice of trade-off function ψ, the allocation done by the algorithm is completely determined. In particular, once we fix π and ψ, the number of bidders of each type x_i gets determined. Let $\mathbf{a} = [\alpha_1, \ldots, \alpha_{k-1}]$ be the vector denoting these values *i.e.* $x_i = \alpha_i$.

For every instance π and function ψ we define a new LP $L(\pi, \psi)$ as follows:

$$\max \ \mathbf{c} \cdot \mathbf{x} \ \ s.t. \ \ \mathbf{A}\mathbf{x} \leq \mathbf{l} \ \ \mathbf{x} \geq \mathbf{0}$$

where $\mathbf{A}, \mathbf{c}$ are same as in LP L and $\mathbf{l} = \mathbf{A}\mathbf{a}$,

Note that LP $L(\pi, \psi)$ is simply a relaxed tautology! Moreover, the right-hand side of the constraints have unknown quantities. $L(\pi, \psi)$ by itself is not very useful, however we get some insight by considering its dual $D(\pi, \psi)$.

$$\min \ \mathbf{l} \cdot \mathbf{y} \ \ s.t. \ \ \mathbf{A}^\top \mathbf{y} \geq \mathbf{c} \ \ \mathbf{y} \geq \mathbf{0}$$

Compare LP $D(\pi, \psi)$ with the LP D from the previous formulation:

$$\min \ \mathbf{b} \cdot \mathbf{y} \ \ s.t. \ \ \mathbf{A}^\top \mathbf{y} \geq \mathbf{c} \ \ \mathbf{y} \geq \mathbf{0}$$

The only difference between $D(\pi, \psi)$ and D is the objective function. The constraints remain the same. Hence, any feasible solution to D should also be a feasible solution to $D(\pi, \psi)$. Hence, $\mathbf{y}^*$, also serves as a feasible solution to $D(\pi, \psi)$. Moreover, $\mathbf{a}$ and $\mathbf{y}^*$ are obtained by setting all inequalities to equalities. Clearly, they satisfy complementary slackness conditions, and are therefore optimal solutions to $L(\pi, \psi)$ and $D(\pi, \psi)$, respectively.

Claim 2.3.1. *For any instance π and a monotonically decreasing function trade-off function ψ, $\mathbf{y}^*$ is an optimal solution to $D(\pi, \psi)$.*

We cannot say the same thing about L and $L(\pi, \psi)$ since the constraints are different. But we seek to relate the right hand side of the inequalities. The assumptions we made to simplify the analysis of Balance still hold good. They are:

- The budget is divided into k slabs and each bidder spends money from ith slab before moving to slab $i + 1$th slab.
- A bidder is of type j if at the end of algorithm, he has spent between $((j-1)/k, j/k]$ fraction of his budget.
- Each bidder of type j has spent exactly a fraction of j/k.

As before, let β_i denote the total money spent by bidders from slab i. Hence, $\beta_1 = N/k$, $\beta_i = N/k - (\alpha_1 + \cdots + \alpha_{i-1})/k$.

Let $\Delta(\pi, \psi) = \mathbf{l} - \mathbf{b}$ We know $\mathbf{l} = \mathbf{A}\mathbf{a}$, so the ith component of $\mathbf{l}$ is

$$\alpha_1(1 + \frac{i-1}{k}) + \alpha_2(1 + \frac{i-2}{k}) + \cdots + \alpha_i$$

The ith component of $\mathbf{b}$ is iN/k. Hence, $l_i - b_i$ is:

$$\begin{aligned} l_i - b_i &= (\alpha_1 + \cdots + \alpha_i) + \frac{(i-1)}{k}\alpha_1 + \cdots + \frac{(i-2)}{k}\alpha_2 + \cdots + \frac{1}{k}\alpha_{i-1} - \frac{iN}{k} \\ &= (\alpha_1 + \cdots + \alpha_i) + \left(\frac{iN}{k} - \sum_{j=1}^{i} \beta_j\right) - \frac{iN}{k} \\ &= (\alpha_1 - \beta_1) + \cdots + (\alpha_i - \beta_i) \end{aligned}$$

Claim 2.3.2.

$$\mathbf{l} = \mathbf{b} + \Delta(\pi, \psi)$$

where $\Delta(\pi, \psi)$ is a $k-1$ dimensional vector with the i component being $(\alpha_1 - \beta_1) + \cdots + (\alpha_i - \beta_i)$

Proof. By definition of $\Delta(\pi, \psi)$. □

Now we compare the performance of our algorithm ALG with the optimal algorithm OPT. We do so by comparing the two algorithms by the way they treat a query q. Let us define ALG(q) and OPT(q) as the revenue earned our algorithm and optimal algorithm, respectively. Say a query q is of type i if OPT assigns it to a bidder of type i. Say a query q lies in slab i if ALG pays for it from slab i. For the example we considered earlier(Fig.2.1), type(q)=1 since OPT gave the query to bidder 1, and slab(q)=1 since our algorithm paid for q from slab 1.

Claim 2.3.3. *For each query q such that $1 \leq type(q) \leq k-1$*

$$OPT(q)\psi(type(q)) \leq ALG(q)\psi(slab(q))$$

Proof. Let us suppose OPT assigned the query q to a bidder b. Since b belongs to type less than k, he is still currently bidding from some slab $j \leq \text{type}(q)$ when q arrives. Since our algorithm assigns the query to the bidder with the maximum bid $\times$ ψ(bidder's slab), the inequality follows. □

Claim 2.3.4.

$$\sum_{i=1}^{k-1} \psi(i)(\alpha_i - \beta_i) \leq \frac{N}{k}$$

Proof.

$$\sum_{q:\text{type}(q)=i} \text{OPT}(q) = \alpha_i$$
$$\sum_{q:\text{slab}(q)=i} \text{ALG}(q) = \beta_i$$

$$\begin{aligned}\sum_{q:\text{type}(q)\leq k-1} \text{OPT}(q)\psi(\text{type}(q)) &= \sum_{i=1}^{k-1} \sum_{q:\text{type}(q)=i} \text{OPT}(q)\psi(i) \\ &= \sum_{i=1}^{k-1} \psi(i)\alpha_i\end{aligned}$$

and

$$\begin{aligned}\sum_{q:\text{type}(q)\leq k-1} \text{ALG}(q)\psi(\text{slab}(q)) &\leq \sum_{q:\text{type}(q)\leq k} \text{ALG}(q)\psi(\text{slab}(q)) \\ &\leq \sum_{q:\text{type}(q)\leq k-1} \text{ALG}(q)\psi(\text{slab}(q)) + \frac{N}{k} \\ &= \sum_{i=1}^{k-1} \sum_{q:\text{slab}(q)=i} \text{ALG}(q)\psi(i) + \frac{N}{k} \\ &= \sum_{i=1}^{k-1} \psi(i)\beta_i + \frac{N}{k}\end{aligned}$$

From Claim.2.3.3,

$$\sum_{q:\text{type}(q)\leq k-1} [\text{OPT}(q)\psi(\text{type}(q)) - \text{ALG}(q)\psi(\text{slab}(q))] \leq 0$$

$$\therefore \sum_{i=1}^{k-1} \psi(i)(\alpha_i - \beta_i) \leq \frac{N}{k}$$

□

Now we use the observation that we made earlier: $\mathbf{y}^*$ is the optimal solution to the dual $D(\pi, \psi)$. The trick is to choose the trade-off function as a function of optimal solution $\mathbf{y}^*$ itself.

Claim 2.3.5. *For the function ψ_k defined as*

$$\psi_k(i) = \sum_{j=i}^{k-1} y_j^* = 1 - \left(1 - \frac{1}{k}\right)^{k-i+1}$$

the competitive ratio of the algorithm is $(1 - \frac{1}{e})$, as k tends to infinity.

Proof. The optimal solution to the LP $L(\pi, \psi)$ has the value $\mathbf{l} \cdot \mathbf{y}^*$. We show that this value cannot be too large and the theorem follows.

We know that $\mathbf{c} \cdot \mathbf{x}^* \leq N/e$ from analysis of BALANCE and $\mathbf{b} \cdot \mathbf{y}^* \leq \mathbf{c} \cdot \mathbf{x}^*$. Hence, $\mathbf{b} \cdot \mathbf{y}^* \leq N/e$.

$$\mathbf{l} \cdot \mathbf{y}^* = (\mathbf{b} + \boldsymbol{\Delta})\mathbf{y}^* \leq N/e + \boldsymbol{\Delta} \cdot \mathbf{y}^*$$

$$\begin{aligned}
\boldsymbol{\Delta} \cdot \mathbf{y}^* &= \sum_{i=1}^{k-1} y_i^*((\alpha_1 - \beta_1) + \cdots + (\alpha_i - \beta_i)) \\
&= \sum_{i=1}^{k-1} (\alpha_i - \beta_i)(y_i^* + \cdots + y_{k-1}^*) \\
&= \sum_{i=1}^{k-1} (\alpha_i - \beta_i)\psi(i) && \text{from our choice of } \psi \\
&\leq \frac{N}{k} && \text{from Claim. 2.3.4}
\end{aligned}$$

As k tends to infinity, the competitive ratio tends to $(1 - 1/e)$. □

Direct proof

In the above analysis we derived both the competitive ratio and the trade-off function together. Once we have the correct trade-off function ψ, the analysis of the competitive ratio is simpler. The proof is as follows: From the definition of α and β variables, we have

$$\forall\, i: \quad \beta_i = \frac{N - \sum_{j=1}^{i-1} \alpha_j}{k}$$

From Claim 2.3.4 we have,

$$\sum_{i=1}^{k-1} \psi(i)(\alpha_i - \beta_i) \leq \frac{N}{k}$$

where the trade-off function ψ is:

$$\psi(i) = 1 - \left(1 - \frac{1}{k}\right)^{k-i+1}$$

From the above relations we can show[1]:

$$\sum_{i=1}^{k} \alpha_i \frac{k-i+1}{k} \leq \frac{N}{e}$$

The left hand side of the inequality is the amount that is unspent at the end of the algorithm. This proves that the competitive ratio is $1 - 1/e$.

Removing assumptions

We now remove two assumptions we made earlier in the analysis of the algorithm:

- All bidders have unit budgets
- The optimal solution exhausts all the money from each bidder

Assume that bidder i has a budget of B_i. Let us say that the current type of a bidder during the run of algorithm is j if he has spent between $(j-1)/k$ and j/k fraction of his budget at that time. Our algorithm now assigns the next query to the bidder who maximizes the product of his bid and ψ(current type). We define α and β similar to earlier definition. Define β_i^j to be the amount spent by the bidder j from the interval $[\frac{i-1}{B}_j, \frac{i}{k}B_j)$ of his budget. Let $\beta_i = \sum_{j=1}^{k} \beta_j^i$. Let α_i be the amount of money that the optimal allocation gets from the bidders of finaly type i. Let $\alpha = \sum_i \alpha_i$, be the total revenue obtained in the optimal allocation. The proof of the competitive ratio changes minimally, the only change is the relation involving β. In the direct proof described above we now have:

$$\forall\, i: \quad \beta_i \geq \frac{\alpha - \sum_{j=1}^{i} \alpha_j}{k}$$

$$\sum_{i=1}^{k-1} \psi(i)(\alpha_i - \beta_i) \leq \frac{N}{k}$$

We these set of relations, we can show that the money left at the end of the algorithm is less than $\frac{\alpha}{e}$ and hence prove that the competitive ratio is atleast $1 - 1/e$.

[1] I was not able to verify this claim

Chapter 3

Conclusion

The algorithms in this study are quite typical of online algorithms in general: simple to state but difficult to analyze. In the second chapter, we have looked at Internet ad auctions more as a generalized version of bipartite matching and therefore made no use of many context-specific assumptions that can exploited to obtain better revenue. There are lot of results concerning maximization of revenue for ad auctions which make such assumptions. Some of the aspects that typically arise in online auctions are:

- *Distribution of queries.* In practice, there is lot of statistical information available about queries. The distribution of queries also vary according to time of the day, special events, etc.
- *Gaming.* Gaming by the advertisers is a serious concern in auctions. A search engine usually does not charge the entire bid amount from the highest bidder, but rather a small amount more than the second highest bidder. A bidder can deplete a competitor bid by bidding slightly short of the winning bid. Once the budget of the competitor is exhausted, the query can be obtained at a smaller bid. The algorithm we have studied ignores these issues. Designing a *truthful* mechanism for auctions is a hard problem in general and a topic of active research.

Both algorithms RANKING and BALANCE, which achieve a competitive ratio of $1 - 1/e$, are optimal in the sense that no randomized algorithm can have a competitive ratio better than $1 - 1/e$ for both problems. Tight examples which restrict the ratio of any randomized algorithm can be found in [3] and [4].

Many online algorithms in general make use of trade-off functions, but technique of using ensemble of LPs to derive the optimal trade-off function is quite unique to this problem. Understanding the scope and applicability of this technique remains open.

References

[1] Benjamin E. Birnbaum and Claire Mathieu. On-line bipartite matching made simple. *SIGACT News*, 39(1):80–87, 2008.

[2] Bala Kalyanasundaram and Kirk Pruhs. An optimal deterministic algorithm for online b-matching. *Theor. Comput. Sci*, 233(1-2):319–325, 2000.

[3] Karp, Vazirani, and Vazirani. An optimal algorithm for on-line bipartite matching. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1990.

[4] Aranyak Mehta, Amin Saberi, Umesh V. Vazirani, and Vijay V. Vazirani. Adwords and generalized online matching. *J. ACM*, 54(5), 2007.