Derandomisation

We will study a randomised algorithm for a problem, which can be derandomised attaining the same approximation performance as the randomised version. The algorithm also allows me to use Hoeffding's Inequality, another strong probability bound. The book does not use it, but introduces it through two problems (4.6, 4.7) at the end of Chapter 4.

Hoeffding's Inequality. Let $Y_1, \ldots, Y_n$ be independent random variables with bounded support $[\ell_i, u_i]$, $i = 1, \ldots, n$, and mean values $E[Y_i]$. Then for $Y = \sum_{i=1}^{n} Y_i$, we have $E[Y] = \sum_{i=1}^{n} E[Y_i]$ and for $\delta > 0$,

$$Pr\{|Y - E[Y]| > \delta\} \leq 2e^{-2\delta^2 / \sum_{i=1}^{n} (u_i - \ell_i)^2}.$$

In particular for i.i.d. random variables

$$Pr\{|Y - E[Y]|| > \delta\} \leq 2e^{-2\delta^2 / n(u-\ell)^2}.$$

The Set Balancing Problem

Given a $n \times n$ matrix A all of whose entries are 0 or 1 find a n-dimensional column vector b consisting of entries -1 or 1 such as to minimise

$$||Ab||_\infty = \max_i \{a_i^T b\}$$

with a_i the i-th row of A. The simple randomised algorithm proposed sets each element of b to -1 or $+1$ with probability $1/2$ independently.

If $a_{ij} = 0$ then obviously $E[a_{ij} b_j] = 0$.
If $a_{ij} = 1$ then $E[a_{ij} b_j] = \frac{1}{2} \cdot (-1) + \frac{1}{2} \cdot 1 = 0$.
Therefore,

$$E[a_i b] = \sum_{j=1}^{n} E[a_{ij} b_j] = 0.$$

Let us first study

$$Pr\{|\sum_{j=1}^{n} a_{ij} b_j| \leq 4\sqrt{n \ln n}\}$$

by bounding the probability of the complementary event

$$Pr\{|\sum_{j=1}^{n} a_{ij} b_j| > 4\sqrt{n \ln n}\}.$$

Using Hoeffding's Inequality gives us

$$Pr\{|\sum_{j=1}^{n} a_{ij} b_j| > \delta\} \leq 2e^{-2\delta^2 / (m_i \cdot 2^2)},$$

where m_i is the number of non-zero elements in row a_i. Since this is bounded by n we have

$$Pr\{|\sum_{j=1}^{n} a_{ij}b_j| > \delta\} \leq 2e^{-\delta^2/2n},$$

Setting $\delta = 4\sqrt{n \ln n}$ yields

$$\begin{aligned} Pr\{|\sum_{j=1}^{n} a_{ij}b_j| > 4\sqrt{n \ln n}\} &\leq 2e^{-16n \ln n/2n} \\ &= 2e^{-8 \ln n} \\ &= 2e^{-\ln n^8} \\ &= \frac{2}{n^8}. \end{aligned}$$

This way we even get a stronger result than in the book, where they prove $Pr\{|\sum_{j=1}^{n} a_{ij}b_j| > 4\sqrt{n \ln n}\} \leq \frac{2}{n^2}$.

Exercise Set Balancing. Prove the statement in the book using Chernoff's bound.

As a result, with our analysis we obtain

$$\begin{aligned} Pr\{||Ab||_\infty > 4\sqrt{n \ln n}\} &\leq \sum_{i=1}^{n} Pr\{|\sum_{j=1}^{n} a_{ij}b_j| > 4\sqrt{n \ln n}\} \\ &\leq n \cdot \frac{2}{n^8} = \frac{2}{n^7}. \end{aligned}$$

Hence,

$$Pr\{||Ab||_\infty \leq 4\sqrt{n \ln n}\} \geq 1 - \frac{2}{n^7}.$$

The method of conditional probabilities

Suppose that with certainty we would like to have a vector b such that $||Ab||_\infty \leq 4\sqrt{n \ln n}$. We will now *derandomise* the algorithm to achieve this. In fact for many problems the best deterministic algorithm is obtained by derandomising a randomised algorithm. There are essentially two methods for derandomisation. I will treat here the method of conditional probabilities.

We can regard the randomised algorithm that we used for Set-Balancing as choosing at random any of the 2^n paths in a computation tree: each level i of the tree corresponds to assigning the value $+1$ or -1 to coordinate b_i of the vector b. This process can be represented by a binary tree, where on a path from the root to a node at level i the values of $b_1, \ldots, b_i$ are set and the values for $b_{i+1}, \ldots, b_n$ are still to be decided. such a node gets than two children, one of them sets $b_{i+1} = +1$ and the other $b_{i+1} = -1$. So we get a binary tree with

2^n leaves. In the leaves all values have been set: a path from the root to a leaf corresponds to a particular solution b.

We call a leaf of this tree GOOD if corresponding solution b has $||Ab||_\infty \leq 4\sqrt{n \ln n}$. Otherwise we call it BAD. We know that, starting from the root, the randomised algorithm reaches a GOOD leaf with probability at least $1 - 2/n^7$. This implies that there exists at least one GOOD leaf.

Now, for any other node a in the tree, let $P(a)$ be the probability that starting from a the randomised algorithm will arrive at a BAD leaf.

Let c and d denote the nodes that are the two children of a in the tree. Then clearly,

$$P(a) = \frac{P(c) + P(d)}{2}$$

and hence

$$P(a) \geq \min\{P(c), P(d)\}.$$

We know that $P(root) \leq 2/n^7$ and having arrived at a leaf we have $P(leaf \text{ is } bad) \in \{0, 1\}$.

Therefore, by selecting at every node the child with the smaller probability of arriving at a BAD leaf must lead to a leaf that has probability 0 of being BAD, hence probability 1 of being a GOOD leaf. If this is rather surprising to you, then think about it for a little longer to convince yourself.

Usually, the obstacle for derandomisation is the computation of the probabilities. It can hardly ever be done efficiently. However, one can often make efficiently estimates of the probabilities. If these estimates are pessimistic we will see that the derandomisation method still works.

Let $\mathcal{E}_i$ be the event that $|a_i b| > 4\sqrt{n \ln n}$, i.e., a failure caused by the i-th row of A, and $\mathcal{E}$ the event that $||Ab||_\infty > 4\sqrt{n \ln n}$. Let $Pr\{\mathcal{E}_i \mid a\}$ denote the conditional probability that $\mathcal{E}_i$ occurs, given that we are in node a. Then,

$$P(a) = Pr\{\mathcal{E} \mid a\} = Pr\{\cup_{i=1}^n \mathcal{E}_i \mid a\} \leq \sum_{i=1}^n Pr\{\mathcal{E}_i \mid a\}.$$

Let us denote this $\sum_{i=1}^n Pr\{\mathcal{E}_i \mid a\}$ by $\hat{P}(a)$. So $P(a) \leq \hat{P}(a)$, and that is why we call $\hat{P}(a)$ a pessimistic estimate of $P(a)$.

We have already shown that

$$Pr\{\mathcal{E}\} = Pr\{\cup_{i=1}^n \mathcal{E}_i \mid root\} \leq \frac{2}{n^7}.$$

From which it is easy to show, and in fact we already did, that

$$\hat{P}(root) \leq 2/n^7.$$

You are asked to make Problem 5.11 to show that

a) $\hat{P}(a) \geq \min\{\hat{P}(c), \hat{P}(d)\}$ for children c and d of a;

b) $\hat{P}(a)$ can be computed in polynomial time.

Since we need to compute at most $2n$ pessimistic probabilities, this deterministic algorithm is a polynomial time algorithm that determines a b such that $||Ab||_\infty \leq 4\sqrt{n \ln n}$.

The Probabilistic Method

We have just seen an example of *the probabilistic method*. This is a tool for proving existential theorems in combinatorial optimisation by the means of randomisation. It is based on two complementary ideas of the same nature:

a) If a random variable has expected value $E[X] = a$, then certainly there exists a realisation of X with value $\geq a$ and a realisation of X with value $\leq a$. (If $E[X] \leq a$ then for sure there exists a realisation of $X \leq a$.)

b) If a random object drawn from some universe of objects has a certain property with non-zero probability then there must exist an object with that property in this universe.

In Set-Balancing we had that with positive probability, and actually not so small probability, a randomly drawn vector b will have $||Ab||_\infty \leq 4\sqrt{n \ln n}$. Thus, using idea b) above this immediately implies the theorem:

Theorem. Given any $0,1$-matrix A for Set-Balancing there always exists a vector b such that $||Ab||_\infty \leq 4\sqrt{n \ln n}$.

I also briefly went through the example of the existence of an $(n, 18, 1/3, 2)$-expander in 5.3 in [B&T], not written out here in my lecture notes.

We have seen more examples where the probabilistic method could have been applied to arrive at existential theorems.

In the Game Tree Evaluation for binary trees we found a randomised algorithm that has an expected number of leaf-nodes to be read of 3^n. Hence, using idea a) above, this implies immediately

Theorem. Every Game Tree can be evaluated by reading only 3^n of the leaf-nodes.

We started the course with the extremely simple example of a randomised algorithm that made in expectation at least 1/2 of the clauses true in any instance of MAXIMUM SATISFIABILITY. Hence, again using idea a),

Theorem. For every instance of MAXIMUM SATISFIABILITY a TRUE-FALSE assignment to the variables exists that makes at least half of the clauses true.

I will close my part by continuing on MAXIMUM SATISFIABILITY and giving a randomised algorithm with a better performance ratio than the 1/2 of the simple one. It is a beautiful application of the randomised rounding technique.

Randomised rounding for MAX-SAT

The new randomised algorithm was given by Goemans and Williamson, and we denote it therefor by GW. Whereas we denote the former simple one by RA. We will show that

$$\frac{E[m^{GW}]}{m^{OPT}} \geq 3/4.$$

Remember that if in MAXIMUM SATISFIABILITY we have a clause with k literals, then RA, setting all variables independently to TRUE or FALSE with probability 1/2, gives a probability of 2^{-k} of not satisfying that clause. Thus, for instances of MAXIMUM SATISFIABILITY without single literal clauses RA already attains the approximation ratio of GW. That is, clauses of size 1 cause the trouble for RA.

To avoid this trouble GW starts by an Integer Linear Programming formulation of the problem. We assume we n variables and m clauses. Introduce for each clause C_j a variable z_j, having value $z_j = 1$ if C_j is satisfied and value $z_j = 0$ otherwise. For each boolean variable x_i we introduce decision variable y_i, with $y_i = 1$ if $x_i =$TRUE and $y_i = 0$ otherwise.

For each clause C_j we define

$$\begin{aligned} C_j^+ &= \{i \mid x_i \text{ appears in } C_j\}; \\ C_j^- &= \{i \mid \neg x_i \text{ appears in } C_j\}. \end{aligned}$$

Then the problem is formulated as

$$\begin{aligned} \max \quad & W = \textstyle\sum_{j=1}^{m} z_j \\ \text{s.t.} \quad & \textstyle\sum_{i \in C_j^+} y_i + \sum_{i \in C_j^-} (1 - y_i) \geq z_j, \quad j = 1, \ldots, m \\ & y_i \in \{0,1\}, \qquad\qquad i = 1, \ldots, n \\ & z_j \in \{0,1\}, \qquad\qquad j = 1, \ldots, m. \end{aligned}$$

Suppose the LP-relaxation of this problem is solved by the solution $(\hat{y}, \hat{z})$ with value W^{LP}. Then we apply randomised rounding by setting independently for each i,

$$Pr\{x_i = \text{True}\} = \hat{y}_i \quad \text{and} \quad Pr\{x_i = \text{False}\} = 1 - \hat{y}_i.$$

Let us compute the probability that clause C_j with k literals is satisfied. W.l.o.g. we may assume that C_j consists of the first k boolean variables $x_1, \ldots, x_k$ in unnegated form, corresponding to the inequality

$$y_1 + y_2 + \cdots + y_k \geq z_j.$$

Let $Z_j = 1$ denote the event that C_j is satisfied. Then

$$\begin{aligned} Pr\{Z_j = 1\} &= Pr\{\cup_{i=1}^{k} \{x_i = \text{True}\}\} \\ &= 1 - \Pi_{i=1}^{k} Pr\{x_i = \text{False}\} \\ &= 1 - \Pi_{i=1}^{k} (1 - \hat{y}_i). \end{aligned}$$

We will bound this probability from below.

Notice that we have, by LP-feasibility, $\sum_{i=1}^{k} \hat{y}_i \geq \hat{z}_j$. Hence we are interested in

$$\begin{aligned} \min \quad & 1 - \Pi_{i=1}^{k} (1 - \hat{y}_i) \\ \text{s.t.} \quad & \textstyle\sum_{i=1}^{k} \hat{y}_i \geq \hat{z}_j \\ & \hat{y}_i \geq 0, \ i = 1, \ldots, n. \end{aligned}$$

It is easily verified that the minimum is attained by $\hat{y}_i = \hat{z}_j / k$, $i = 1, \ldots, k$. Therefore,

$$Pr\{Z_j = 1\} \geq 1 - (1 - \frac{\hat{z}_j}{k})^k.$$

The function $f(x) = 1 - (1 - \frac{x}{k})^k$ is a concave function. Therefore, on any interval $[a, b]$ it is higher than the linear function going through the point $(a, f(a))$ and $(b, f(b))$. In particular $f(x) \geq g(x) = cx + d$ on $[0, 1]$, with $g(0) = f(0) = 1 - (1 - \frac{0}{k})^k = 0$ and $g(1) = f(1) = 1 - (1 - \frac{1}{k})^k$. Hence, inserting into $g(x) = cx + d$, $d = 0$ and $c = 1 - (1 - \frac{1}{k})^k$ yields

$$f(x) = 1 - (1 - \frac{x}{k})^k \geq (1 - (1 - \frac{1}{k})^k) x.$$

Hence,

$$Pr\{Z_j = 1\} \geq \left(1 - (1 - \frac{1}{k})^k\right) \hat{z}_j.$$

Call the randomised rounding algorithm RR and let us compare this probability of RR with the probability achieved by the simple RA.

k	$1-2^{-k}$	$1-(1-\frac{1}{k})^k$
1	0.500	1.000
2	0.750	0.750
3	0.875	0.704
$\vdots$	$\vdots$	$\vdots$

Thus, in expectation RR works perfectly for clauses of size 1, is equal to the simple RA on clauses of size 2 and is outperformed by RA on clauses of size greater than 2.

The obvious idea is now implemented in the GW algorithm: run RA, then run RR and take the best of the two solutions. (In fact, one could even randomise this step by flipping a fair coin to decide on using RA or RR.) Let n_1 be the expected number of TRUE clauses produced by RA and n_2 the expected number of TRUE clauses produced by RR.

Theorem.

$$W^{GW} = \max\{n_1, n_2\} \geq \frac{3}{4}\sum_{j=1}^{m} \hat{z}_j.$$

This automatically leads to the desired result since $\frac{3}{4}\sum_{j=1}^{m} \hat{z}_j = \frac{3}{4}W^{LP} \geq \frac{3}{4}W^{OPT}$.

Proof of Theorem. Clearly

$$\max\{n_1, n_2\} \geq \frac{n_1 + n_2}{2}.$$

$$n_1 = \sum_{k=1}^{n} \sum_{\{j|C_j \text{ has } k \text{ literals}\}} (1-2^{-k}) \geq \sum_{k=1}^{n} \sum_{\{j|C_j \text{ has } k \text{ literals}\}} (1-2^{-k})\hat{z}_j;$$

$$n_2 = \sum_{k=1}^{n} \sum_{\{j|C_j \text{ has } k \text{ literals}\}} (1-(1-\frac{1}{k})^k)\hat{z}_j.$$

Thus,

$$\frac{n_1 + n_2}{2} \geq \sum_{k=1}^{n} \frac{(1-2^{-k}) + (1-(1-\frac{1}{k})^k)}{2} \sum_{\{j|C_j \text{ has } k \text{ literals}\}} \hat{z}_j$$

and it is easy to verify that, for all k,

$$(1-2^{-k}) + (1-(1-\frac{1}{k})^k) \geq 3/2$$

implying the result. □.

Exercises

Chapter 4: Exercise 4.7, Problem 4.15

Chapter 5: Problems 5.3, 5.11.
Open Problems 5.7, 5.10, 5.14

Material

During the fourth lecture I treated:
Sections 4.1, and 5.1,5.3 (but not 5.3.1),5.6 [MR]