DM582 Solutions

Mads Anker Nielsen
madsn20@student.sdu.dk

March 12, 2024

This document contains written solution to exercise problems from the course DM582 (spring 2024). The solutions given here might differ from the solutions discussed in class. In class, we place more emphasis on the intuition leading to the correct answer. Please do not consider reading these solutions an alternative to attending the exercise classes.

References to CLRS refer to the book *Introduction to Algorithms, 4th edition* by Cormen, Leiserson, Rivest, and Stein.

References to KT refer to the book *Algorithm Design, 1st edition* by J. Kleinberg and E. Tardos.

This document will inevitably contain mistakes. If you find some, please report them to me (Mads) so that I can correct them.

Sheet 4

KT, Exercise 6

Exercise

One of the (many) hard problems that arises in genome mapping can be formulated in the following abstract way. We are given a set of n markers $\{\mu_1, \ldots, \mu_n\}$ — these are positions on a chromosome that we are trying to map — and our goal is to output a linear ordering of these markers. The output should be consistent with a set of constraints, each specified by a triple (μ_i, μ_j, μ_k), requiring that μ_j lie between μ_i and μ_k in the total ordering that we produce. (Note that this constraint does not specify which of μ_i or μ_k should come first in the ordering, only that μ_j should come between them.)

Now it is not always possible to satisfy all constraints simultaneously, so we wish to produce an ordering that satisfies as many as possible. Unfortunately, deciding whether there is an ordering that satisfies at least k' of the k constraints is an NP-complete problem (you don't have to prove this).

Give a constant $\alpha > 0$ (independent of n) and an algorithm with the following property. If it is possible to satisfy k^* of the constraints, then the algorithm produces an ordering of markers satisfying at least αk^* of the constraints. Your algorithm may be randomized; in this case, it should produce an ordering for which the expected number of satisfied constraints is at least αk^*.

Suggested solution

We give an algorithm which satisfies $\frac{1}{3}k$ constraints in expectation. The algorithm is very simple: simply output a random permutation of the markers. We show that this indeed results in satisfying a third of the constraints in expectation.

For $i \in [k]$ let X_i be an indicator random variable for the event that the i-th constraint is satisfied. Then $X = \sum_{i=1}^{k} X_i$ is a random variable whose value is the number of constraints satisfied. Now,

$$E[X_i] = P[i\text{-th constraint satisfied}] = \frac{1}{3}$$

since all relative orders of the three markers are equally likely. By linearity of expectation

$$E[X] = \sum_{i=1}^{k} E[X_i] = \frac{1}{3}k$$

which is what we wanted to show.

KT, Exercise 7

Exercise

In Section 13.4, we designed an approximation algorithm to within a factor of 7/8 for the MAX 3-SAT Problem, where we assumed that each clause has terms associated with three different variables. In this problem, we will consider the analogous MAX SAT Problem: Given a set of clauses $C_1, ..., C_k$ over a set of variables $X = \{x_1, \ldots, x_n\}$,find a truth assignment satisfying as many of the clauses as possible. Each clause has at least one term in it, and all the variables in a single clause are distinct, but otherwise we do not make any assumptions on the length of the clauses: There may be clauses that have a lot of variables, and others may have just a single variable.

(a) First consider the randomized approximation algorithm we used for MAX 3-SAT, setting each variable independently to true or false with probability 1/2 each. Show that the expected number of clauses satisfied by this random assignment is at least $k/2$, that is, at least half of the clauses are satisfied in expectation. Give an example to show that there are MAX SAT instances such that no assignment satisfies more than half of the clauses.

(b) If we have a clause that consists only of a single term (e.g., a clause consisting just of x_1, or just of x_2), then there is only a single way to satisfy it: We need to set the corresponding variable in the appropriate way. If we have two clauses such that one consists of just the term x_i, and the other consists of just the negated term x_i, then this is a pretty direct contradiction. Assume that our instance has no such pair of "conflicting clauses"; that is, for no variable x_i do we have both a clause $C = \{x_i\}$ and a clause $C = \{\overline{x_i}\}$. Modify the randomized procedure above to improve the approximation factor from 1/2 to at least .6. That is, change the algorithm so that the expected number of clauses satisfied by the process is at least .6k.

(c) Give a randomized polynomial-time algorithm for the general MAX SAT Problem, so that the expected number of clauses satisfied by the algorithm is at least a .6 fraction of the maximum possible.

Suggested solution

(a) Let X_i be an indicator random variable for the event that C_i is satisfied. Then $X = \sum_{i=1}^{n} X_i$ is a random variable whose values is the number of satisfied clauses. A clause with l literals is false iff all literals are false, and each literal is set to false independently and with probability $1/2$. Hence, the probability that a clause is false is $(1/2)^l$ which is at most $1/2$ when $l \geq 1$. Thus,

$$E[X_i] \geq \left(1 - \frac{1}{2}\right) = \frac{1}{2}.$$

By linearity of expectation

$$E[X] = \sum_{i=1}^{k} E[X_i] \geq \frac{1}{2}k$$

which is what we wanted.

The instance $x_1 \wedge \overline{x_1}$ shows that this bound is tight. That is, no algorithm can guarantee satisfaction of more than half of the clauses in the general case.

(b) We modify the algorithm by increasing the probability of satisfying clauses with a single literal. If x_i ($\overline{x_i}$) is a literal of a single clause, we set x_i to true (false) with probability $6/10$ such that the probability of satisfying the clause is $6/10$. This is well-defined since we have no contradictory single clauses. If a literal does not occur in a single clause, we set it to true with probability $\frac{1}{2}$ and false with probability $\frac{1}{2}$.

Again, let X_i be an indicator random variable for the event that C_i is satisfied. Then we have $E[X_i] = \frac{6}{10} = 0.6$ if C_i contains a single literal and $E[X_i] \geq 1 - \left(\frac{6}{10}\right)^l > 0.6$ for clauses with $l \geq 2$ literals. Hence, we have $E(X) = \sum_{i=1}^{k} E(X_i) \geq 0.6k$ as desired.

Note: the choice of 0.6 **is somewhat arbitrary. The same analysis works if we replace** 0.6 **by** $\phi - 1 \approx 0.618$ **where** ϕ **is the golden ratio.**

(c) We start by modifying the SAT instance in the following way. For each single literal clause $C = \{x_i\}$, if the clause $C' = \{\overline{x_i}\}$ also occurs in the SAT instance, remove C'. Suppose there are k' clauses left after removing such clauses. Now, apply the algorithm above. This gives an assignment which satisfies at least $0.6k'$ clauses in expectation. Since it is not possible to satisfy more than k' clauses, we have that the expected number of clauses satisfied by the algorithm is at least a 0.6 fraction of the maximum possible.

Exercise from course webpage

Exercise

This problem concerns a randomized algorithm for coloring graphs. Assume we have a graph, $G = (V, E)$, and colors, $\{1, 2, 3, 4\}$. Then $f\colon V \to \{1, 2, 3, 4\}$ is called a 4-coloring of G. We say that an edge $uv \in E$ is *good* with respect to f if $f(u) \neq f(v)$.

1. Suppose that each vertex independently gets color i with probability $1/4$ for $i \in \{1, 2, 3, 4\}$. Let the random variable X be the number of edges in E that are good with respect to the coloring f. Show that $E[X] = \frac{3|E|}{4}$.
2. Use the result above and the probabilistic method to conclude that every graph has a 4-coloring f such that at least $\frac{3|E|}{4}$ of its edges are good with respect to f.
3. Define a randomized algorithm that, for a given graph $G = (V, E)$, finds a 4-coloring f^* of V such that at least $\frac{3|E|}{4}$ of the edges of G are good with respect to f^*.
4. Follow the construction from MAX 3-SAT to compute a bound on the expected running time of your algorithm?

Suggested solution

1. For $e \in E$ let X_e be an indicator random variable for the event that e is good. Now $X = \sum_{e \in E} X_e$. For any $uv \in E$, uv is good iff u is assigned a color different from $f(v)$, which occurs with probability $\frac{3}{4}$. Thus
$$E[X_e] = \frac{3}{4}$$
for any $e \in E$ and by linearity of expectation
$$E[X] = \sum_{e \in E} \frac{3}{4} = \frac{3}{4}|E|$$
as desired.
2. In general, for a sample space S and random variable $X : S \to \mathbb{R}$ there must be some outcome $s \in S$ such that $X(s) \geq E[X]$. We give a short proof by contradiction. Let S be any sample space and let

X be a random variable on S. Suppose there is no $s \in S$ such that $X(s) \geq E[X]$ (i.e. $X(s) < E[X]$ for all $s \in S$). By the definition of expectation

$$\begin{aligned} E[X] &= \sum_{s \in S} X(s)P[s] \\ &< \sum_{s \in S} E[X]P[s] \\ &= E[X] \sum_{s \in S} P[s] \\ &= E[X], \end{aligned}$$

so $E[X] < E[X]$ which is a contradiction.

The claim from the task follows since $E[X] = \frac{3}{4}|E|$ and thus there exists some outcome f (a particular 4-coloring from the sample space of all possible 4-colorings) such that $X(f)$ (the number of good edges wrt. f) is at least $\frac{3}{4}|E|$.

3. We simply generate random colorings until we find a coloring f^* such that at least $\frac{3}{4}|E|$ edges are good wrt. f^*.

4. Let f be a random coloring and let p denote the probability that at least $\frac{3}{4}$ edges are good wrt. f. We obtain a lower bound on p.

   Again, let X be a random variable whose value is the number of good edges. From part 3. we know that $E[X] = \frac{3}{4}|E|$. Now, by the definition of expectation

   $$\frac{3}{4}|E| = E[X] = \sum_{i=0}^{|E|} iP[X = i].$$

   Let c be the largest natural number which is strictly smaller that $\frac{3}{4}|E|$. That is, c is the largest number of edges that can be good without $\frac{3}{4}|E|$ edges being good.

   Our goal is to bound $p = \sum_{i=c+1}^{|E|} P[X = i]$ from below using the above

equality. We see that

$$\begin{aligned}
\frac{3}{4}|E| &= \sum_{i=0}^{|E|} iP[X=i] \\
&= \sum_{i=0}^{c} iP[X=i] + \sum_{i=c+1}^{|E|} iP[X=i] \\
&\leq \sum_{i=0}^{c} cP[X=i] + \sum_{i=c+1}^{|E|} |E|P[X=i] \\
&= c\sum_{i=0}^{c} P[X=i] + |E| \sum_{i=c+1}^{|E|} P[X=i] \\
&= c(1-p) + |E|p \\
&\leq c + |E|p
\end{aligned}$$

so $|E|p \geq \frac{3}{4}|E| - c \geq \frac{1}{4}$ by the choice of c and thus $p \geq \frac{1}{4|E|}$.

Thus, the expected number of trails until we find a coloring satisfying at least $\frac{3}{4}|E|$ edges is $4|E|$.

Note. You can think of the fact that $\frac{3}{4}|E| - c \geq \frac{1}{4}$ as follows: if $3|E|$ happens to be divisible by 4, then $\frac{3}{4}|E| - c = 1$, and if $3|E|$ is not divisible by 4, it is at least 1 above a multiple of 4 and thus $\frac{1}{4}3|E| - c \geq \frac{1}{4}$.

KT, Exercise 2

Exercise

Revisit this exercise to also find some upper bound on the probability of at least 1000 Democrats voting for the R candidate.

Suggested solution

Let X be the number of Democrats voting for the R candidate (X is a random variable). We know from previously that $E[X] = \frac{1}{100} \cdot 80000 = 800$. Now, by the definition of expectation

$$\begin{aligned}
800 = E[X] &= \sum_{i=0}^{80000} iP[X = i] \\
&= \sum_{i=0}^{999} iP[X = i] + \sum_{i=1000}^{80000} iP[X = i] \\
&\geq \sum_{i=1000}^{80000} iP[X = i] \\
&\geq 1000 \sum_{i=1000}^{80000} P[X = i] \\
&= 1000P[X \geq 1000]
\end{aligned}$$

which implies $P[X \geq 1000] \leq \frac{800}{1000} = \frac{4}{5}$.

One can generalize the above derivation to obtain that $P[X \geq a] \leq \frac{E[X]}{a}$ for any positive real a and any non-negative random variable X. This is known as Markov's inequality.

KT, Exercise 8

Exercise

Let $G = (V, E)$ be an undirected graph with n nodes and m edges. For a subset $X \subseteq V$, we use $G[X]$ to denote the subgraph induced on X — that is, the graph whose node set is X and whose edge set consists of all edges of G for which both ends lie in X.

We are given a natural number $k \leq n$ and are interested in finding a set of k nodes that induces a "dense" subgraph of G; we'll phrase this concretely as follows. Give a polynomial-time algorithm that produces, for a given natural number $k \leq n$, a set $X \subseteq V$ of k nodes with the property that the induced subgraph $G[X]$ has at least $\frac{mk(k-1)}{n(n-1)}$ edges. You may give either (a) a deterministic algorithm, or (b) a randomized algorithm that has an expected running time that is polynomial, and that only outputs correct answers.

Suggested solution

We give a very simple randomized algorithm: pick a set X of k vertices uniformly at random among all k-subsets of V until we obtain a set X such that $G[X]$ has at least $\frac{k(k-1)}{n(n-1)}m$ edges. Call $X \subseteq V$ with $|X| = k$ *good* if $G[X]$ contains at least $\frac{k(k-1)}{n(n-1)}m$ edges.

We start by showing that the expected number of edges in $G[X]$ is $\frac{k(k-1)}{n(n-1)}m$. Using this, we derive a lower bound on the probability that X is good.

For $e \in E$ let I_e be an indicator random variable for the event that $e \in E(G[X])$. Then $I = \sum_{e \in E} I_e$ is a random variable whose value is the number of edges in $G[X]$. Now, $E[I_{uv}] = P[uv \in E(G[X])] = P[u \in X \text{ and } v \in X] = \frac{k(k-1)}{n(n-1)}$ and by linearity of expectation

$$E[I] = \sum_{e \in E} E[I_e] = \frac{k(k-1)}{n(n-1)}m.$$

Let p be the probability that X is good. And let c be the largest natural number which is strictly smaller that $\frac{k(k-1)}{n(n-1)}m$. That is, c is the largest number of edges that can be in $G[X]$ without X being good. We also observe that $I \leq \binom{k}{2}$ since $G[X]$ contains k vertices. We now obtain a lower bound on p.

We see that

$$\begin{aligned}
E[I] = \frac{k(k-1)}{n(n-1)}m &= \sum_{i=0}^{m} iP[I=i] \\
&= \sum_{i=0}^{\binom{k}{2}} iP[X=i] \\
&= \sum_{i=0}^{c} iP[X=i] + \sum_{i=c+1}^{\binom{k}{2}} iP[X=i] \\
&\leq \sum_{i=0}^{c} cP[X=i] + \sum_{i=c+1}^{\binom{k}{2}} \binom{k}{2} P[X=i] \\
&= c(1-p) + \binom{k}{2} p \\
&\leq c + \binom{k}{2} p
\end{aligned}$$

so $\binom{k}{2}p \geq \frac{k(k-1)}{n(n-1)}m - c \geq \frac{1}{n(n-1)}$. Thus, $p \geq \frac{2}{k(k-1)n(n-1)}$ which yields an expected running time of $\frac{k(k-1)n(n-1)}{2} \in O((kn)^2)$.

Exercise from course webpage

Exercise

For a graph, $G = (V, E)$, a *spanning bipartite subgraph* G' of G is defined by a partition (V_1, V_2) of V, and the edges $E' \subseteq E$ that have an endpoint in both parts.

Consider the following randomized algorithm for finding a spanning bipartite subgraph of an arbitrary graph: Independently, for each vertex $v \in V$, decide uniformly at random if vertex v is in V_1 or V_2.

1. Give a lower bound on the expected number of edges m' in E' as a function of $m = |E|$.
2. How can you use your result to conclude that any graph has a spanning bipartite subgraph with $m' \geq m/2$?
3. Design a deterministic, polynomial-time algorithm for this problem, finding a spanning bipartite subgraph G' of any graph G, where $m' \geq m/2$.

Suggested solution

1. For $e \in E$ let X_e be an indicator random variable for the event that $e \in |E'|$. Then $X = \sum_{e \in E} X_e$ is a random variable whose value is the number of edges m' in the resulting bipartite graph. An edge $uv \in E'$ iff u and v are in different parts of the partition (V_1, V_2), which happens with probability $\frac{1}{2}$. Thus,

$$E[X_e] = \frac{1}{2}$$

for any $e \in E$ and by linearity of expectation

$$E[X] = \sum_{e \in E} \frac{1}{2} = \frac{1}{2}m.$$

2. Since the expected number of edges in a bipartite graph obtained from a random partition is $\frac{1}{2}m$, there must be some partition (V_1, V_2) such that the number of edges in the induced bipartite graph is at least $\frac{1}{2}m$.
3. The following algorithm achieves this:

(i) Initially, set $V_1 = V$ and $V_2 = \emptyset$.

(ii) Repeatedly pick $v \in V_i$ for some $i \in [2]$ such that v has more than half its neighbors in V_i and move v to V_{3-i}.

(iii) When no such vertex v exists, return (V_1, V_2).

We observe that when moving a vertex v from V_i to V_{3-i}, the number of edges in G' only increases by the choice of v. Thus, the given procedure is indeed an algorithm (it terminates).

When the algorithm terminates, $d_{G'}(v) \geq \frac{1}{2} d_G(v)$ for all $v \in V$ and thus

$$\begin{aligned} m' &= \frac{1}{2} \sum_{v \in V} d_{G'}(v) \\ &\geq \frac{1}{2} \sum_{v \in V} \frac{1}{2} d_G(v) \\ &= \frac{1}{4} \sum_{v \in V} d_G(v) \\ &= \frac{1}{2} m \end{aligned}$$

where we use that $\sum_{v \in V(H)} d_H(v) = 2|E(H)|$ for any graph H.[1]

[1]For a graph H, $d_H(v)$ denotes the degree of the vertex v in H