ShanghaiTech University

CS240 Algorithm Design and Analysis

Spring 2022

Problem Set 4

Due: 23:59, June 7, 2022

1. Submit your solutions to Gradescope (www.gradescope.com).
2. In "Account Settings" of Gradescope, set your FULL NAME to your Chinese name.
3. If you want to submit a handwritten version, scan it clearly.
4. When submitting your homework in Gradescope, match each of your solution to the corresponding problem number.
5. Late homeworks submitted within 24 hours of the due date will be marked down 25%, submitted within 48 hours will be marked down 50%, and will not be accepted past 48 hours.
6. You are required to follow ShanghaiTech's academic honesty policies. You are allowed to discuss problems with other students, but you must write up your solutions by yourselves. You are not allowed to copy materials from other students or from online or published resources. Violating academic honesty can result in serious sanctions.

Problem 1:

Let DOUBLE-SAT=$\{\psi\ |\psi$ is a Boolean formula with at least two satisfying assignments$\}$. Show that deciding whether a formula is in DOUBLE-SAT is NP-complete.

Solution:

(1) DOUBLE-SAT $\in$NP: For any two different assignments to all variables, we can verify whether each clause is satisfied in both cases in polynomial time.

(2) Reduction of 3-SAT to DOUBLE-SAT: Given a 3cnf-function ϕ, create a new Boolean function $\phi^{'}$ by adding a new clause $(x \vee \neg x)$ to ϕ, where x is a new variable not in ϕ. Then check if $< \phi^{'>\in}$ DOUBLE-SAT. Also, the reduction clearly works in polynomial time.

(3) DOUBLE-SAT problem $\phi^{'}$ is satisfied in and only if the 3-SAT problem ϕ is satisfied:

$\Rightarrow$: If $\phi^{'}$ is satisfied, there is one assignment to variables except x at least, so the ϕ is satisfied too.

$\Leftarrow$: If ϕ is satisfied, there are two assignments in which x=0 and x=1 to satisfy $\phi^{'}$. so the $\phi^{'}$ is satisfied too.

Thus, DOUBLE-SAT is NPC.

Problem 2:

Suppose you are given a set of n items with sizes $s_1, \ldots, s_n$, and an infinite number of bins, each of capacity $C \geq \max_i s_i$. You want to pack the items into as few bins as possible. That is, you want to partition the items into k subsets, such that the total size of the items in each subset is $\leq C$, and make k as small as possible. Give a polynomial time 2-approximation algorithm for this problem and show that your algorithm is correct.

Solution:

Arrange the bins in sequence and place each item into the first bin in which it fits.

Prove: Suppose k bins needed for the above algorithm, and the number of optimal solution is k^*.

When k=1, there must be $k^* = 1$, $k < 2 * k^*$.

If $k > 1$: we consider any two bins, the total size in the two bins $> C$ (because we put the item into other bins when we there is no enough space in the bin for this item). So we can get $s_1 + s_2 + ... + s_n > k * \frac{C}{2}$. We also have $k^* * C \geq s_1 + s_2 + ... + s_n$. Thus, $k < 2 * k^*$

Problem 3:

Recall that in the *maximum matching* problem, you are given an undirected unweighted graph $G = (V, E)$, and want to choose the largest possible set of edges which do not touch each other. That is, you want to find the largest $E' \subseteq E$ such that for all $e_1, e_2 \in E'$, e_1 and e_2 do not share a common vertex. While there are polynomial time algorithms for this problem, they are somewhat slow. Give a simple, fast greedy algorithm for this problem which achieves a 2-approximation.

Hint: Think about the 2-approximation algorithm for vertex cover we discussed in class.

Solution:

Algorithm: Randomly choose an edge first, and then perform the next loop: (1) take a random edge (actually in order it was given); (2) if this edge meets requirements then add, or execute the next cycle;

Finally we get a matching.

Prove: Suppose M^* is a maximal matching in the graph G, and M be the matching returned by our approximation algorithm (obviously this algorithm returns a valid matching).

For all e$\in M$, let M_e^* be the set of edges in M^* who have common vertices with e, meaning all $e_{M^*} \in M^*$ have $e_{M^*} \cap e \neq 0$. Obviously $\forall e \in M : |M_e^*| \leq 2$, since at the worst, e=(u,v) and both u,v are matched to some other vertex in M^*.

We know that $M^* = \cup_{e \in M} M_e^*$(if there exists $e \in M^*$ with no common vertices with any $e^{' \in M}$, the algorithm would add e to M), So we have: $|M^*| \leq \sum_{e \in M} |M_e^*| \leq 2|M|$.

Problem 4:

Suppose that Alice is trying to send a k bit message to Bob over a noisy channel. Each bit that Alice sends is lost with $\frac{1}{2}$ probability. Alice can detect the bit is lost and resend it, but the bit may again be lost. Show that Alice can send $O(k + \log(1/\epsilon))$ bits in total and guarantee that Bob receives all k bits with probability at least $1 - \epsilon$.

Solution:

Let X_i be an indicator random variable, where $X_i = 1$ if Bob receives the i-th message successfully. Then, after k successful messages, Bob will receive all the bits. Let $n = 4k + 16\ln(1/\epsilon)$ and $X = \sum_{i=1}^{n} X_i$, we have $\mu = \mathrm{E}[X]$ and $\mu = 2k + 8\ln(1/\epsilon)$. If we find that $\Pr[X \leq \mu/2] \leq \epsilon$, then we can know that there are at least $\mu/2 \geq k$ successful messages with probability at least $1 - \epsilon$. Since all the messages are independent, we can apply a Chernoff Bound:

$$\begin{aligned} Pr[X \leq (1-1/2)\mu] &\leq e^{-\mu(1/2)^2(1/2)} \\ &\leq e^{-\mu/8} \\ &\leq e^{-(2k+8\ln(1/\epsilon))/8} \\ &\leq e^{-\ln(1/\epsilon)} \\ &\leq \epsilon \end{aligned}$$

Problem 5:

Suppose we want to build a counter, which has initial value 1. The counter has two functions Inc and Count. Each call to Inc increases the counter's value by 1, and each call to Count returns the counter's current value. If Inc is called at most n times, it is easy to build a counter using $O(\log n)$ bits.

The *Morris counter* is a randomized algorithm for an approximate counter, which uses less memory than a standard counter. The algorithm uses a single integer value c, initialized to 0. Each time Inc is called, we increase c by 1 with probability $\frac{1}{2^c}$. When Count is called, we return $2^c - 1$. Suppose Inc is called n times and then we call Count; let t be Count's return value. Morris proved that $\mathrm{E}(t) = n$ and $\mathrm{Var}(t) \leq n^2$. Thus, the Morris counter returns an unbiased estimate of the count, but has high variance.

We can reduce the variance of the Morris counter by running multiple copies of it in parallel and combining their outputs in some way. Describe this algorithm in detail. How many copies do you need to guarantee your algorithm returns a $(1 \pm \epsilon)$-approximation of the correct count with probability $\geq \frac{3}{4}$?

Solution:

We can run x independent copies of the Morris counter and return the average value.

Denote A as the average value of x copies of the Morris Counter. Each copy of the Morris Counter will return a value with $\mathrm{E}(t) = n$, so we know that $E[A] = n$. Also, the variance of the Morris Counter is $\mathrm{Var}(t) \leq n^2$, we can know that $\mathrm{Var}[A] = x \cdot \mathrm{Var}[t]/x^2 \leq n^2/x$.

By Chebychev's Inequality, we know that:

$$\begin{aligned}\Pr[|A - n| \geq \frac{2n}{\sqrt{x}}] &\leq \frac{\mathrm{Var}[A]}{(2n/\sqrt{x})^2} \\ &\leq \frac{n^2}{x}\frac{x}{4n^2} \\ &\leq \frac{1}{4}\end{aligned}$$

Therefore, we find that $A = n(1 \pm 2/\sqrt{x})$ with probability $\geq \frac{3}{4}$. Then, by choosing $x = 4/\epsilon^2$, we conclude that $A = n(1 \pm \epsilon)$ with probability $\geq \frac{3}{4}$.