INFO 6205
Program Structure and Algorithms
Assignment 6 Solutions

Student Name: ________________________________________
Professor: Nik Bear Brown

Q1 (20 Points) Suppose you're acting as a consultant for the Port Authority of a small Pacific Rim nation. They're currently doing a multi-billion-dollar business per year, and their revenue is constrained almost entirely by the rate at which they can unload ships that arrive in the port.

Here's a basic sort of problem they face. A ship arrives, with *n* containers of weight *w*1, *w*2, . . . , *wn*. Standing on the dock is a set of trucks, each of which can hold *K* units of weight. (You can assume that *K* and each *wi* is an integer.) You can stack multiple containers in each truck, subject to the weight restriction of *K*; the goal is to minimize the number of trucks that are needed in order to carry all the containers. This problem is NP-complete (you don't have to prove this).

A greedy algorithm you might use for this is the following. Start with an empty truck, and begin piling containers 1, 2, 3, . . . into it until you get to a container that would overflow the weight limit. Now declare this truck "loaded" and send it off; then continue the process with a fresh truck. This algorithm, by considering trucks one at a time, may not achieve the most efficient way to pack the full set of containers into an available collection of trucks.

(a) Give an example of a set of weights, and a value of *K*, where this algorithm does not use the minimum possible number of trucks.

(b) Show, however, that the number of trucks used by this algorithm is within a factor of 2 of the minimum possible number, for any set of weights and any value of *K*.

Solution: Suppose you're acting as a consultant for the Port Authority of a small Pacific Rim nation. They're currently doing a multi-billion-dollar business per year, and their revenue is constrained almost entirely by the rate at which they can unload ships that arrive in the port.

Here's a basic sort of problem they face. A ship arrives, with *n* containers of weight $w_1$, $w_2$,…, $w_n$. Standing on the dock is a set of trucks, each of which can hold *K* units of weight. (You can assume that *K* and each $w_i$ is an integer.) You can stack multiple containers in each truck, subject to the weight restriction of *K*; the goal is to minimize the number of trucks that are needed in order to carry all the containers. This problem is NP-complete (you don't have to prove this).

A greedy algorithm you might use for this is the following. Start with an empty truck, and begin piling containers 1, 2, 3,… into it until you get to a container that would overflow the weight limit. Now declare this truck "loaded" and send it off; then continue the process with a fresh truck. This algorithm, by considering trucks one at a time, may not achieve the most efficient way to pack the full set of containers into an available collection of trucks.

(a) Give an example of a set of weights, and a value of *K,* where this algorithm does not use the minimum possible number of trucks.

(b) Show, however, that the number of trucks used by this algorithm is within a factor of 2 of the minimum possible number, for any set of weights and any value of *K.*

Q2 (20 Points) You are asked to consult for a business where clients bring in jobs each day for processing. Each job has a processing time $t_i$ that is known when the job arrives. The company has a set of ten machines, and each job can be processed on any of these ten machines.

At the moment the business is running the simple Greedy-Balance Algorithm we discussed in Section 11.1 of K&T. They have been told that this may not be the best approximation algorithm possible, and they are wondering if they should be afraid of bad performance. However, they are reluctant to change the scheduling as they really like the simplicity of the current algorithm: jobs can be assigned to machines as soon as they arrive, without having to defer the decision until later jobs arrive.

In particular, they have heard that this algorithm can produce solutions with makespan as much as twice the minimum possible; but their experience with the algorithm has been quite good: They have been running it each day for the last month, and they have not observed it to produce a makespan more than 20 percent above the average load.

Solution You are asked to consult for a business where clients bring in jobs each day for processing. Each job has a processing time $t_i$ that is known when the job arrives. The company has a set of ten machines, and each job can be processed on any of these ten machines.

At the moment the business is running the simple Greedy-Balance Algorithm we discussed in Section 11.1. They have been told that this may not be the best approximation algorithm possible, and they are wondering if they should be afraid of bad performance. However, they are reluctant to change the scheduling as they really like the simplicity of the current algorithm: jobs can be assigned to machines as soon as they arrive, without having to defer the decision until later jobs arrive.

In particular, they have heard that this algorithm can produce solutions with makespan as much as twice the minimum possible; but their experience with the algorithm has been quite good: They have been running it each day for the last month, and they have not observed it to produce a makespan more than 20 percent above the average load, $\frac{1}{10}\sum_i t_i$

To try understanding why they don't seem to be encountering this factor-of-two behavior, you ask a bit about the kind of jobs and loads they see. You find out that the sizes of jobs range between 1 and 50, that is, $l \leq t_i \leq 50$for all jobs i; and the total load $\sum_i t_i$ is quite high each day: it is always at least 3,000.

Prove that on the type of inputs the company sees, the Greedy-Balance. Algorithm will always find a solution whose makespan is at most 20 percent above the average load

To try understanding why they don't seem to be encountering this factor-of-two behavior, you ask a bit about the kind of jobs and loads they see. You find out that the sizes of jobs range between 1 and 50, that is, $1 \leq ti \leq 50$ for all jobs $i$; and the total load $\sum_i t_i$ is quite high each day:
it is always at least 3,000.
Prove that on the type of inputs the company sees, the Greedy-Balance Algorithm will always find a solution whose makespan is at most 20 percent above the average load.

Q3 (20 Points) Consider the problem of finding a stable state in a Hopfield neural network, in the special case when all edge weights are positive. This corresponds to the Maximum-Cut Problem that we discussed earlier in the chapter: For every edge *e* in the graph *G,* the endpoints of *G* would prefer to have opposite slates.

Now suppose the underlying graph G is connected and bipartite; the nodes can be partitioned into sets *X* and *Y* so that each edge has one end in *X* and the other in *Y.* Then there is a natural "best" configuration for the Hopfield net, in which all nodes in *X* have the state +1 and all nodes in *Y* have the state –1. This way, all edges are *good,* in that their ends have opposite states.

The question is: In this special case, when the best configuration is so clear, will the State-Flipping Algorithm described in the text [as long as there is an unsatisfied node, choose one and flip its state) always find this configuration? Give a proof that it will, or an example of an input instance, a starting configuration, and an execution of the State-Flipping Algorithm that terminates at a configuration in which not all edges are good.

is to minimize the overall cost $\sum_{s \in A} f_s + \sum_{u \in U} \min_{s \in A} d_{us}$. Give an $H(n)$ approximation for this problem.

(Note that if all service costs $d_{us}$ are 0 or infinity, then this problem is exactly the Set Cover Problem: $f_s$ is the cost of the set named *s,* and $d_{us}$ is 0 if node *u* is in set *s,* and infinity otherwise.)

Solution The state-flipping algorithm will not always find this configuration

For example, Let *C* he a graph consisting of a cycle of length four: there are nodes

$v_2$, $v_s$, $v_4$ and edges ($v_i$, $v_2$), ($v_2$, $v_s$), ($v_s$, $v_4$), ($v_4$, $v_{ii}$). Then if we start the state-flipping algorithm in a configuration where nodes vi and v2 have state +1, and nodes va and $v_4$ have state —1, then no improving *move is possible.*

Q4 (20 Points) 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*; well 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.

Solution First we give an algorithm that produces a subgraph whose expected number of edges has the desired value. For this, we simply choose *t*! nodes uniformly at random from G. Now, for i < *j,* let $X_{tia}$ be a random variable equal to 1 if there is an edge between our ith and jth node choices, and equal to 0 otherwise.

Of the n(rt — 1) *choices* for i and *j,* t.here are 2m. that, yield **AM** edge (since **as** edge (9L, can be chosen either by picking a in position *i* and v in position *j,* or by picking v in position i and u in position *j).* Thus *E [X•jj =* $^2$*nxF,*

The expected number of edges we get in total is

E *E [4] = (*$_2{}^k$*)* $\underline{on^27}$

We now want to turn this into an algorithm with expected polynomial running time, which always produces a subgraph with at least this many edges. The analogous issue came up with MAX 3-SAT, and we use the same idea here: For this we use the same idea as in the analogous MAX 3-SAT: we run the above randomized algorithm repeatedly until it produces a subgraph with at least the desired number of edges.

Let p+ be the probability that one iteration of this succeeds; our overall running time will be the (polynomial) time for one iteration, tunes 1/p+. First note that the maximum number of edges we can find is e = E%=$^{11}$, and we're seeking e' = e..,$^2$7,$^7$%). Let e" denote the greatest integer strictly less than *e'.* Let

$p_i$ denote the probability that we find a subgraph with exactly *j* edges. Thus *pi" = p<sub>i</sub>;* we define p- E, *p<sub>i</sub> = pt.* Then we have

c' =

= E E *jpa*

E epi + *epi*

i<e'*j>e*$^{1}$

— ell(1 — *p+)* + ($^{k}{}_{2}$)P+

from which it follows that

(e." + ($_{2}$))/)$^{+}$____e e"$_{7}$,($_{7}$, _ 1)$^{\cdot}$

Since e" < (2), we have p+and so we are done.

Q5 (20 Points) . Consider a balls-and-bins experiment with 2*n* balls but only two bins. As usual, each ball independently selects one of the two bins, both bins equally likely. The expected number of balls in each bin is *n.* In this problem, we explore the question of how big their difference is likely to be. Let $X_1$ and $X_2$ denote the number of balls in the two bins, respectively. ($X_1$ and $X_2$ are random variables.) Prove that for any $\varepsilon > 0$ there is a constant c > 0 such that the probability $\Pr[X_1 - X_2 > c\sqrt{n}] \leq \varepsilon$.

Solution The mean for $X_2$ is n, so in order to have *X,* — $X_2$ > *c,rn,* we need **X2 <** *E [X<sub>2</sub>]*— =

*(1 — 6) F, [X<sub>2</sub>1*for =Plugging this into the Chernoff lower hound, the probability this happens is

*e-i6$^{2}$E[X2) = e* $^{c2'4}$

This can be made smaller than a constant *c* by choosing the undetermined constant *c* large enough.