Note: Your TA may not get to all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. The discussion worksheet is also a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 Coffee Shops

A rectangular city is divided into a grid of $m \times n$ blocks. You would like to set up coffee shops so that for every block in the city, either there is a coffee shop within the block or there is one in a neighboring block. (There are up to 4 neighboring blocks for every block). It costs r_{ij} to rent space for a coffee shop in block ij.

Write an integer linear program to determine which blocks to set up the coffee shops at, so as to minimize the total rental costs.

(a) What are your variables, and what do they mean?

(b) What is the objective function?

(c) What are the constraints?

(d) Solving the non-integer version of the linear program gets you a real-valued solution. How would you round the LP solution to obtain an integer solution to the problem? Describe the algorithm in at most two sentences.

(e) What is the approximation ratio obtained by your algorithm?

(f) Briefly justify the approximation ratio.

Solution:

(a) There is a variable for every block x_{ij}, i.e., $\{x_{ij} | i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\}\}$. This variable corresponds to whether we put a coffee shop at that block or not.

(b) $\min \sum_{i=1}^{m} \sum_{j=1}^{n} r_{ij} x_{ij}$. Alternatively, $\min t$ is correct as well as long as the correct constraint is added.

(c) (i) $x_{ij} \geq 0$: This constraint just corresponds to saying that there either is or isn't a coffee shop at any block. $x_{ij} \in \{0, 1\}$ or $x_{ij} \in \mathbb{Z}_+$ is also correct.

(ii) For every $1 \leq i \leq m, 1 \leq j \leq n$:

$$x_{ij} + x_{(i+1),j}\mathbb{1}_{\{i+1 \leq m\}} + x_{(i-1),j}\mathbb{1}_{\{i-1 \geq 1\}} + x_{i,(j+1)}\mathbb{1}_{\{j+1 \leq n\}} + x_{i,(j-1)}\mathbb{1}_{\{j-1 \geq 1\}} \geq 1$$

This constraint corresponds to that for every block, there needs to be a coffee shop at that block or a neighboring block.

$\mathbb{1}_{\{i+1 \leq m\}}$ means "1 if $\{i + 1 \leq m\}$, and 0 otherwise". It keeps track of the fact that we may not have all 4 neighbors on the edges, for instance.

(iii) If the objective was $\min t$, then the constraint $\sum_{i=1}^{m} \sum_{j=1}^{n} r_{ij} x_{ij} \leq t$ needs to be added.

(d) Round to 1 all variables which are greater than or equal to 1/5. Otherwise, round to 0.
In other words, put a coffee shop on (i, j) iff $x_{i,j} \geq 0.2$.

(e) Using the rounding scheme in the previous part gives a 5-approximation.

(f) Notice that every constraint has at most 5 variables. So for every constraint, there exists at least one variable in the constraint which has value $\geq 1/5$ (not everyone is below average). The total cost of the rounded solution is at most $5 \cdot$ LP-OPT, since $r_{ij} \leq 5r_{ij}x_{ij}$ for any x_{ij} that gets rounded up, and the other i, j pairs contribute nothing to the cost of the rounded solution. Since Integral-OPT $\geq$ LP-OPT (the LP is more general than the ILP), our rounding gives value at most 5 LP-OPT $\leq$ 5 Integral-OPT. So we get a 5-approximation.

2 Local Search for Max Cut

Sometimes, local search algorithms can give good approximations to NP-hard problems. In the Max-Cut problem, we have an unweighted graph $G(V, E)$ and we want to find a cut (S, T) with as many edges "crossing" the cut (i.e. with one endpoint in each of S, T) as possible. One local search algorithm is as follows: Start with any cut, and while there is some vertex $v \in S$ such that more edges cross $(S - v, T + v)$ than (S, T) (or some $v \in T$ such that more edges cross $(S + v, T - v)$ than (S, T)), move v to the other side of the cut. Note that when we move v from S to T, v must have more neighbors in S than T.

(a) Give an upper bound on the number of iterations this algorithm can run for (i.e. the total number of times we move a vertex).

(b) Show that when this algorithm terminates, it finds a cut where at least half the edges in the graph cross the cut.

Solution:

(a) $|E|$ iterations. Each iteration increases the number of edges crossing the cut by at least 1. The number of edges crossing the cut is between 0 and $|E|$, so there must be at most $|E|$ iterations.

(b) $\delta_{in}(v)$ be the number of edges from v to other vertices on the same side of the cut, and $\delta_{out}(v)$ be the number of edges from v to vertices on the opposite side of the cut. The total number of edges crossing the cut the algorithm finds is $\frac{1}{2}\sum_{v \in V} \delta_{out}(v)$, and the total number of edges in the graph is $\frac{1}{2}\sum_{v \in V}(\delta_{in}(v) + \delta_{out}(v))$. We know that $\delta_{out}(v) \geq \delta_{in}(v)$ for all vertices when the algorithm terminates (otherwise, the algorithm would move v across the cut), so the former is at least half as large as the latter.

3 Modular Arithmetic

(a) Show that if $a_1 \equiv b_1 \pmod{n}$ and $a_2 \equiv b_2 \pmod{n}$, then $a_1 + a_2 \equiv b_1 + b_2 \pmod{n}$.

(b) Show that for integers a_1, b_1, a_2, b_2, and n, if $a_1 \equiv b_1 \pmod{n}$ and $a_2 \equiv b_2 \pmod{n}$, then $a_1 \cdot a_2 \equiv b_1 \cdot b_2 \pmod{n}$.

(c) What is the last digit (i.e., the least significant digit) of 3^{4001}?

Solution:

(a) By the definition of modular arithmetic, there are integers $k_1, \ldots, k_4 \in \{0, \ldots, n-1\}$ are r integers such that $a_1 = k_1 + r_1 n$, $b_1 = k_1 + r_1' n$, $a_2 = k_2 + r_2 n$, and $b_2 = k_2 + r_2' n$. So we get $a_1 + a_2 = k_1 + r_1 n + k_2 + r_2 n = k_1 + k_2 + n(r_1 + r_2) \equiv k_1 + k_2 \pmod{n}$. Likewise: $b_1 + b_2 = k_1 + r_1' n + k_2 + r_2' n = k_1 + k_2 + n(r_1' + r_2') \equiv k_1 + k_2 \pmod{n}$. So $a_1 + a_2 \equiv a_1 \cdot a_2 \pmod{n}$.

(b) Using the same k and r values from the previous part, we get $a_1 \cdot a_2 = k_1(r_2 n) + k_1 k_2 + r_1 n(r_2 n) + r_1 n(k_2) = k_1 k_2 + n(\dots) \equiv k_1 k_2 \pmod{n}$. Likewise $b_1 \cdot b_2 = k_1(r_2' n) + k_1 k_2 + r_1' n(r_2' n) + r_1' n(k_2) = k_1 k_2 + n(\dots) \equiv k_1 k_2 \pmod{n}$. So $a_1 \cdot a_2 \equiv b_1 \cdot b_2 \pmod{n}$.

(c) The last digit of 3^{4001} is the same as the value of $3^{4001} \pmod{10}$. We can find this value through the following computation:

$$3^{4001} \equiv (3^4)^{1000} \cdot 3^1 \equiv (81)^{1000} \cdot 3^1 \equiv 1^{1000} \cdot 3^1 \equiv 3 \pmod{10}$$

4 Random Prime Generation

Lagrange's prime number theorem states that as N increases, the number of primes less than N is $\Theta(N/\log(N))$.

An important primitive in cryptography is the ability to sample a prime number uniformly at random. Assume we can verify that an n-bit number is a prime in $O(n^2)$ time. Briefly describe a randomized algorithm that samples a prime uniformly at random from all primes in $\{2, 3, \dots, 2^n - 1\}$ with expected runtime polynomial in n. What is the expected runtime of your algorithm?

(Recall that if we have a coin that lands heads with probability p, the expected number of coin flips we make before we see the first heads is $1/p$.)

Solution: We repeatedly sample a number from $\{2, 3, \dots, 2^n - 1\}$ uniformly at random, and verify if it is prime or not. If it is, we output it, otherwise we sample a new number. Notice that since we sample from these numbers uniformly at random and resample whenever we get a composite number, this is equivalent to sampling uniformly at random from all the primes.

Of all n-bit numbers, $\Theta(2^n/\log(2^n)) = \Theta(2^n/n)$ are prime. So the probability p of randomly choosing a prime is $\Theta((2^n/n)/2^n) = \Theta(1/n)$. Substituting this value of p in our equation for the expected number of primes we have to sample, we get $E = \Theta(1/(1/n)) = \Theta(n)$. So the expected runtime is $O(n^3)$.

Notice that in this algorithm, the randomness is in the runtime and not the correctness; It always returns a correct answer, but might take a long time to do so. Algorithms of this form are called *Las Vegas Algorithms*.