CS 124 Homework 7: Spring 2022

Your name: Lev Kruglyak

Collaborators: Sahil Kuchlous, Swati Goel, Adam Mohamed

No. of late days used on previous psets: 12
No. of late days used after including this pset: 12^{+}

Homework is due Wednesday at midnight ET. You are allowed up to **twelve** (college)/**forty** (extension school) late days through the semester, but the number of late days you take on each assignme nt must be a nonnegative integer at most **two** (college)/**four** (extension school).

Try to make your answers as clear and concise as possible; style may count in your grades. Assignments must be submitted in pdf format on Gradescope. If you do assignments by hand, you will need to scan in your results to turn them in.

You can collaborate with other students that are currently enrolled in this course in brainstorming and thinking through approaches to solutions but you should write the solutions on your own: you must wait one hour after any collaboration or use of notes from collaboration before any writing in your own solutions that you will submit.

For all homework problems where you are asked to give an algorithm, you must prove the correctness of your algorithm and establish the best upper bound that you can give for the running time. Generally better running times will get better credit; generally exponential time algorithms (unless specifically asked for) will receive no or little credit. You should always write a clear informal description of your algorithm in English. You may also write pseudocode if you feel your informal explanation requires more precision and detail, but keep in mind pseudocode does NOT substitute for an explanation. Answers that consist solely of pseudocode will receive little or not credit. Again, try to make your answers clear and concise.

Problem 1. Consider the problem MAX-k-CUT, which is like the MAX CUT problem, except that we partition the vertices into k disjoint sets, and we want to maximize the number of edges between sets.

1. **(10 points)** Give a deterministic algorithm that finds a partition within a factor of $1 - \frac{1}{k}$ of optimal.
2. **(5 points)** Give a randomized algorithm that's a generalization of the randomized algorithm for MAX CUT from class that finds a partition that, in expectation, is within a factor of $1 - \frac{1}{k}$ of optimal.

(a) Consider the following algorithm: Our solution will be an assignment of every vertex with a number between 1 and k; these correspond to the partitioning of the vertices into k groups. We start by initializing every vertex to 1, then iterating over the vertices, we check if changing its number to any other number would increase the number of edges between sets. (we can check its neighbors locally to see if the global number of edges between sets increases.) Repeat this until no number changes, i.e. the system stabilizes.

There are a few things we must check to prove correctness. First of all, since this algorithm is in the form of an infinite loop, we must check that the terminating condition is always met so the algorithm doesn't go on indefinitely. Recall that the upper bound on a maximum k-cut is equal to the number of edges in the graph. Every iteration that changes a vertex number will increase the number of k-cut edges by at least 1. Thus, after $|E|$ iterations where a change has been made, the loop is guaranteed to terminate.

Next, let's assume that after the algorithm terminates with some near-optimal assignment of the vertices. We claim that the resulting k-cut is within a factor of $1 - \frac{1}{k}$ of the optimal solution. Recall that the upper bound on a maximum k-cut is $|E|$. Suppose we had some arbitrary vertex v with N edges, and assume for the sake of contradiction that in our assignment, the number of adjacent edges to v which do not cross the k-cut is greater than N/k. This means that there are less than $\frac{k-1}{k}N$ edges of v in the k-cut. There are $k-1$ other number assignment types so by the Pigeonhole principle, there must be some group g such that the number of edges from v to a vertex in a group g is less than N/k. This implies that moving v to g would increase the size of the cut, which is a contradiction. Thus at least $\frac{k-1}{k} = 1 - \frac{1}{k}$ of every vertices' edges must be in the cut, so at least $\left(1 - \frac{1}{k}\right)|E|$ edges must be in the cut. $|E|$ is an upper bound on the optimal solution, so our solution is within $1 - \frac{1}{k}$.

Next, let's analyze the performance of the algorithm. For any vertex v, the time complexity to check which group it would be best to move it to is $O(|E(v)|$ where $E(v)$ is the set of adjacent edges to v. Every iteration, we do this for all $|V|$ vertices, so we iterate over every edge at most twice, giving us a runtime of $O(|E|)$. Since we potentially repeat this $|E|$ times, the total runtime is $O(|E|^2)$.

(b) The random version of this algorithm is quite straightforward: for each vertex, we assign it a random number between 1 and k inclusive. We claim that the expected size of the resulting partition is within a factor of $1 - \frac{1}{k}$ of the optimal size. Let e be an arbitrary edge between two vertices v and u. The probability that they were assigned the same number is $k \cdot \frac{1}{k^2} = \frac{1}{k}$, so the probability that e is cut is $1 - \frac{1}{k}$, and the expected number of cuts is $\left(1 - \frac{1}{k}\right)|E|$, which is within a factor of $1 - \frac{1}{k}$ of the upper bound (of the optimal) $|E|$ so we are done.

Evaluating the runtime of this algorithm, if we assume that random number generation is constant time, the runtime of this algorithm is $O(|V|)$ since we simply generate a random number for every vertex.

Problem 2. (15 points) We have considered, in the context of a randomized algorithm for 2SAT, a random walk with a completely reflecting boundary at 0—that is, whenever position 0 is reached, with probability 1 we move to position 1 at the next turn. Consider now a random walk with a partially reflecting boundary at 0– whenever position 0 is reached, with probability 3/4 we move to position 1 at the next turn, and with probability 1/4 we stay at 0. Everywhere else the random walk either moves up or down 1, each with probability 1/2.
Find the expected number of moves to reach n starting from position i using a random walk with a partially reflecting boundary. (You may assume there is a unique solution to this problem, as a function of n and i; you should prove that your solution satisfies the appropriate recurrence.)

As usual with these types of random walk problems, we set up a recurrence relation. Let $T(i)$ be the expected number of moves to reach n starting from position i. The conditions in the problem give us

$$T(i) = \begin{cases} \frac{3}{4}T(1) + \frac{1}{4}T(0) + 1 & i = 0 \\ \frac{1}{2}T(i+1) + \frac{1}{2}T(i-1) + 1 & 0 < i < n \\ 0 & i = n \end{cases}$$

We claim that the equation

$$T(i) = \left(n^2 + \frac{n}{3}\right) - \left(i^2 + \frac{i}{3}\right)$$

is a solution to this recurrence. To verify this, substitute it into the conditions:

$$\begin{aligned} T(n) &= \left(n^2 + \frac{n}{3}\right) - \left(n^2 + \frac{n}{3}\right) = 0 \\ T(0) &= n^2 + \frac{n}{3} = \frac{3}{4}\left(n^2 + \frac{n}{3} - 1 - \frac{1}{3}\right) + \frac{1}{4}\left(n^2 + \frac{n}{3}\right) + 1 \\ &= \frac{3}{4}T(1) + \frac{1}{4}T(0) + 1 \\ T(i) &= \left(n^2 + \frac{n}{3}\right) - \left(i^2 + \frac{i}{3}\right) \\ &= n^2 + \frac{n}{3} - i^2 - 1 - \frac{i-1}{6} - \frac{i+1}{6} + 1 \\ &= \frac{1}{2}\left(n^2 + \frac{n}{3} - (i-1)^2 + \frac{i-1}{3}\right) + \frac{1}{2}\left(n^2 + \frac{n}{3} - (i+1)^2 - \frac{i+1}{3}\right) + 1 \\ &= \frac{T(i-1)}{2} + \frac{T(i+1)}{2} + 1 \end{aligned}$$

This completes the proof, since we are allowed to assume that there is a unique recurrence.

Problem 3. Suppose we have a random walk with boundaries 0 and n, starting at position i. As mentioned in class, this can model a gambling game, where we start with i dollars and quit when we lose it all or reach n dollars. Let W_t be our winnings after t games, where W_t is defined only until we hit a boundary (at which point we stop). Note that W_t is negative if we are at a position j with $j < i$; that is, we have lost money. If the probability of winning and the probability of losing 1 dollar each game are $1/2$, then with probability $1/2$, $W_{t+1} = W_t + 1$ and with probability $1/2$, $W_{t+1} = W_t - 1$. Hence

$$\mathbb{E}[W_{t+1}] = \frac{1}{2}\mathbb{E}[W_t + 1] + \frac{1}{2}\mathbb{E}[W_t - 1] = \mathbb{E}[W_t],$$

where we have used the linearity of expectations at the last step. Therefore when the walk reaches a boundary, the expected winnings is $\mathbb{E}[W_0] = 0$. We can use this to calculate the probability that we finish with 0 dollars. Let this probability be p_0. Then with probability p_0 we lose i dollars, and with probability $1 - p_0$ we gain $n - i$ dollars. Hence

$$p_0(-i) + (1 - p_0)(n - i) = 0,$$

from which we find $p_0 = (n - i)/n$.
Gossip Girl encounters this game in one of her classes at Constance Billard High School. Unfortunately, because it is high school, the game is not fair; instead, the probability of losing a dollar each game is $2/3$, and the probability of winning a dollar each game is $1/3$. Each student starts with i dollars and gets to leave high school if they reach n dollars before going bankrupt. We wish to extend the above argument to this case.

(a) **(5 points)** Show that $\mathbb{E}[2^{W_{t+1}}] = \mathbb{E}[2^{W_t}]$.

(b) **(5 points)** Use this to determine the probability of finishing with 0 dollars and the probability of finishing with n dollars when starting at position i.

(c) **(10 points)** Generalize the argument to handle the case where the probability of losing is $p > 1/2$. (Hint: Try using $\mathbb{E}[c^{W_t}]$ for some constant c.)

(a) From the problem description we have the recurrence:

$$\mathbb{E}[W_{t+1}] = \frac{1}{3}\mathbb{E}[W_t + 1] + \frac{2}{3}\mathbb{E}[W_t - 1].$$

Raising everything to the power of 2, we can thus expand:

$$\begin{aligned}
\mathbb{E}[2^{W_{t+1}}] &= \frac{1}{3}\mathbb{E}[2^{W_t+1}] + \frac{2}{3}\mathbb{E}[2^{W_t-1}] \\
&= \frac{1}{3}\mathbb{E}[2^{W_t} \cdot 2] + \frac{2}{3}\mathbb{E}[2^{W_t}/2] \\
&= \frac{2}{3}\mathbb{E}[2^{W_t}] + \frac{1}{3}\mathbb{E}[2^{W_t}] \\
&= \mathbb{E}[2^{W_t}]
\end{aligned}$$

(b) Suppose we start with i dollars. Let p be the probability that we lose all of our money (i.e. lose i dollars), and let $1 - p$ be the probability that we win all of the money. (i.e.gain $n - i$ dollars) By (a), we have

$$p(2^{-i}) + (1 - p)2^{n-i} = 1.$$

Rearranging terms, we get

$$p = \frac{2^{n-i} - 1}{2^{n-i} - 2^{-i}} = \frac{2^n - 2^i}{2^n - 1}.$$

This is the probability of finishing with zero dollars. Similarly, the probability of finishing with all the money is

$$1 - p = \frac{2^i - 1}{2^n - 1}.$$

Intuitively, both of these forms make sense, as the amount of money you start with increases, the chances that you walk out with more money also increases.

(c) Here, we want the same cancelling behavior as in (a), so we'll set $c = \frac{p}{1-p}$. Then

$$\begin{aligned}
\mathbb{E}[c^{W_{t+1}}] &= (1-p)\mathbb{E}[c^{W_t+1}] + p\mathbb{E}[c^{W_t-1}] \\
&= (1-p)\mathbb{E}[c^{W_t} \cdot c] + \mathbb{E}[c^{W_t}/c] \\
&= (p + (1-p))\mathbb{E}[c^{W_t}] \\
&= \mathbb{E}[c^{W_t}]
\end{aligned}$$

Thus we can use the same trick as in (b); let p be the probability of finishing with zero dollars, $1 - p$ be the probability of finishing with all the money. Then

$$p(c^{-i}) + (1-p)(c^{n-i}) = 1.$$

Expanding, we get

$$p = \frac{c^n - c^i}{c^n - 1} \quad \text{and} \quad 1 - p = \frac{c^i - 1}{c^n - 1}.$$

Problem 4. Patients who require a kidney transplant but do not have a compatible donor can enter a kidney exchange. Usually, the demand for kidneys is far greater than the supply (in the US in 2010, more than 90,000 people were on the wait-list for a transplant, but only 15,000 kidneys were available). Thus, we must decide whom to allocate the kidneys to.
The kidney donation problem has as input a compatibility graph $G(V, E)$, where each patient-donor pair is a vertex, and there is a directed edge $e = (u, v) \in E$ if the donor in u can donate to the patient in v. We wish to maximize the number of transplants.

(a) **(10 points)** Give a (polynomial-time) reduction from the kidney donation problem to an integer linear program. (For this problem, donors are willing to donate a kidney regardless of whether their patient gets a kidney.)

(b) **(5 points)** Suppose that each donor is only willing to donate a kidney if their corresponding patient gets a kidney (so donations form a set of cycles in the graph). Give a (polynomial-time) reduction from this restricted kidney donation problem to an integer linear program.

(c) **(15 points)** Algorithmic matches are not guaranteed to work in practice (due to, e.g., tissue-type incompatibility). Therefore, for each edge e, there is an associated probability f_e that the donation fails. If any of the donations in a cycle fails, the whole cycle fails and no transplants occur. To minimize the issue (and the logistical difficulty of having many transplants occurring at the same time and place), a hospital has decided to cap the length of kidney-donation cycles at 4. Again, each donor is only willing to donate a kidney if their corresponding patient gets a kidney. We wish to maximize the expected number of transplants that succeed. Give a (polynomial-time) reduction from this restricted kidney donation problem to an integer linear program.

(Hint: It may be helpful to calculate the set $C = C_1, C_2, \ldots$ of cycles in G of length at most 4. How big can the set C be?)

(a) The reduction is quite simple, we begin by defining variables $x_1, \ldots, x_{|E|}$, where $|E|$ is the number of edges in the graph. If the solution sets $x_i = 1$ where i is some edge from v to u, we say that the donor in v donates a kidney to the patient in u. We subject the variables to the condition $x_i \in \{0, 1\}$ since every donor either is included or not included. (This is a valid linear programming setup because $0 \leq x_i \leq 1$) Since every donor can only donate one kidney and every patient can receive only one kidney, we add the following conditions for every vertex v:

$$\sum_{i \text{ incoming to } v} x_i \leq 1 \quad \text{and} \quad \sum_{i \text{ outgoing from } v} x_i \leq 1.$$

We then are trying to maximize the quantity

$$\sum_{i \in E} x_i,$$

which corresponds to the total number of kidney's donated/received. This is clearly a polynomial time reduction because there are $|E|$ variables and $2|E| + 2|V|$ conditions.

Let's now prove that this is indeed a valid reduction, i.e. every solution to the kidney problem solves this linear programming problem and vice versa. Suppose we had a solution to the linear programming problem. The conditions clearly ensure that each donor can only donate to once recipient and each recipient can only receive one donation. Then $\sum_{i \in E} x_i$ is equal to the number of donations made, so the linear programming solution is a lower bound on the optimal maximal donation amount.

Conversely, if we had a valid set of kidney donations, notice that this gives us a valid solution to the linear programming problem by the above argument. Then the number of donations made is exactly the quantity we are trying to maximize, so we get a lower bound in the opposite direction. This proves the answers are equal.

(b) In our correspondence between linear programming and kidney donation, this condition translates as

$$\sum_{i \text{ incoming to } v} x_i - \sum_{i \text{ outgoing from } v} x_i = 0$$

for all $v \in V$. This is an addition of $|V|$ conditions so this doesn't change the polynomialness of the reduction. It is also clear to see that any valid kidney donation under this new condition must correspond to a solution to the linear programming, since the added condition ensures that only those donor-patient pairs who donate a kidney can receive one. Then by the same logic in (a), this reduction is valid.

(c) First we calculate a bound on the number of unique cycles of length at most 4. An upper bound on the number of cycles of length ℓ is n^ℓ since every cycle of length ℓ is n^ℓ because every cycle can be identified with the ℓ vertices in the cycle. So the number of cycles of length at most 4 is $O(n^4)$.

Let's then convert the problem into a linear programming problem in the following way. Let C be the set of cycles of length at most 4. Define variables $x_1, \ldots, x_{|C|}$. As before, we add the condition $x_c \in \{0, 1\}$ and for every vertex v we want that at most one selected cycle contains the vertex by adding the condition

$$\sum_{c \text{ contains } v} x_i \leq 1$$

Lastly, for each cycle c let's calculate the probability that the cycle succeeds. It's easy to see that this probability is

$$p_c = \prod_{e \in c} (1 - f_e)$$

where f_e is the probability than an edge succeeds. Then we are trying to maximize

$$\sum_{c \in C} x_c p_c |c|$$

where $|c|$ is the number of vertices/edges in the cycle.

We claim that a valid solution to this linear programming problem corresponds to a valid donation set. Recall that if the linear programming solution sets a $x_c = 1$, we add all of the edges in the cycle to the kidney donor graph. This way, no cycle of length greater than 4 is added to the graph. By the same argument as in the previous part, we also can see that the transplants are comprehensive, so each donor gives to at most one person and each recipient receives at most one kidney. Next note that the function we are optimizing is equal to the expected number of transplants that succeed if we choose the cycles corresponding to the $x_c = 1$. So the solution to the linear program is a lower bound on the maximum expected number of successful transplants.

Conversely, if we have a valid set of kidney donations, for any cycle c of donations we set $x_c = 1$ and zero for all other cycles. Because of the condition that each patient receives one kidney and each recipient can only donate one kidney, the condition $\sum x_c \leq 1$ is satisfied. Then the quantity the linear program maximizes is the expected number of successful transplants as before, so the two answers are equal.

Lastly, we should check that this reduction is polynomial. The only computationally involved part of the reduction is finding all of the cycles of length at most 4. We do this by performing a DFS-variant which is allowed to revisit modified vertices. We run this DFS at every vertex, stopping at a depth of 4 and keeping track of all of the cycles we find. Each run of the DFS takes at most $O(|V|^3)$ so iterating over all vertices takes $O(|V|^4)$. We also add $2c = O(n^4)$ conditions of the form $x_i \in \{0, 1\}$, $|V|$ conditions of the form $\sum x_i \leq 1$ each of length $O(|V|^4)$. The constraint function is also $O(n^4)$ and calculating its terms takes constant time. Since everything here is polynomial, the reduction as a whole is polynomial as well.

Problem 5.

(a) **(14 points)** Consider the TFHiring problem, which takes as input a nonnegative integer k (of TFs to hire) and a list $J = [j_1, j_2, \ldots, j_n]$ of positive integers (lengths, in minutes, of jobs that must be completed each week by some TF—e.g. it may be that $J = [120, 120, 120, 400]$ if the only jobs that need to be done by the TF team are to hold 3 120-minute blocks of office hours and spend 400 minutes creating a pset problem each week). Harvard requires that each TF work at most m minutes per week, for some positive integer m[a]. TFHiring(J, k, m) $= 1$ iff there is a way to assign every job in J to one of k TFs such that that none of the k TFs has to work more than m minutes per week. Show that TFHiring is NP-complete.

(b) **(20 points)** Consider a modified version of the problem: in Discrete-TFHiring, we are given as an additional input a set $T = \{t_1, \ldots t_\ell\}$ for some constant $\ell = |T|$, such that the length of each job in J is in T. That is, in Discrete-TFHiring, we only have ℓ options for the length of each job, whereas in TFHiring, each job could have any integer length.

Give an algorithm to solve Discrete-TFHiring(J, k, m, T) in time $O\left(\frac{nm}{\min(T)}\right)^\ell$.

Hint: Try using dynamic programming. Show that there are $O\left(\frac{m}{\min(T)}\right)^\ell$ distinct sets of jobs that a single TF can do. Show that there are $O(n^\ell)$ distinct subsets of the original set of jobs.

(c) **(10 points)** A $(1+\varepsilon)$-approximation of TFHiring relaxes the constraint that TFs can work at most m minutes per week by allowing some overtime, so that TFs can work at most $(1+\varepsilon)m$ minutes per week. If the jobs can all be assigned with each TF working at most m minutes per week, the output is true; if they can't be assigned even with each TF working up to $(1+\varepsilon)m$ minutes per week, the output is false; if neither of those, the output is allowed to be arbitrary.

Reduce $(1+\varepsilon)$-approximate TFHiring where every job has length at least $m\varepsilon$ to Discrete-TFHiring with $\ell \leq (\frac{1}{\varepsilon})^2$. That is, for every $\varepsilon > 0$, given an input (J, k, m) to TFHiring, turn it into a corresponding input (J', k', m', T) to Discrete-TFHiring such that $|T| \leq (\frac{1}{\varepsilon})^2$ and, if the TFHiring jobs can all be assigned with each TF working at most m minutes per week, the solution to the Discrete-TFHiring you create is true, and if they can't be assigned even with each TF working up to $(1+\varepsilon)m$ minutes per week, the solution to the Discrete-TFHiring problem you create is false.

Hint: Round up the lengths of jobs to the next multiple of c, for some choice of c.

The combination of this part and the previous part gives, for every ε, a $(1+\varepsilon)$-approximation algorithm for TFHiring that runs in time $O(n^{1/\varepsilon^2})$, if every job has length at least $m\varepsilon$.

[a] In real life, $m = 720$, but here m is an arbitrary positive integer input to the problem.

(a) First we show that TFHiring is in NP. Say we are given some assignment of jobs to the TFs. We can easily check if this is valid in polynomial time by first checking that all the jobs are assigned to only one TF and that each TF doesn't have more than m minutes assigned to them.

To show that TFHiring is NP-complete, we will reduce it from the NumberPartition decision problem, which we know is NP-complete. Let $(a_1, \ldots, a_n)$ be an input to NumberPartition with sum $N = \sum a_i$. We can assume N is even, otherwise the answer is always false. Set $m = N/2$ so every TF works at most $N/2$ minutes per week. We then set $k = 2$, and check if 2 TFs are sufficient. It's clear that the answers to these problems are equivalent; we can consider the jobs assigned to a particular TF as the set we split the numbers into. Since every job must be assigned to a TF, this gives us a valid partition. Furthermore, since each TF can have at most $N/2$ minutes assigned, and there are N minutes total, both TFs must have the same number of minutes, corresponding to an equal partition of the sequence. Conversely, any partition of the set into two sets corresponds to a solution of the TF assignment problem under this same correspondence.

(b) We'll use the hint, first we must prove that there are

$$O\left(\frac{m}{\min(T)}\right)^{\ell}$$

distinct sets of jobs that a single TF can do. Say we had some job length t_i. Since a single TF can work at most m minutes per week, they can do at most m/t_i jobs of length t_i. Since $\min(T) \leq t_i$, they can do at most $m/\min(T)$ jobs of length t_i, and this means that for every job length t_i, the TF can do between 0 and $m/\min(T)$ jobs of that length. Furthermore, the number of unique subsets of jobs a single TF can do is at most $(m/\min(T)+1)^{\ell}$ which equals $O(m/\min(T))^{\ell}$.

Next, we need to prove that $O(n^{\ell})$ distinct subsets of the original set of jobs. However this is clear, since there are ℓ job lengths t_i, for each of which there can be at most n jobs.

We are now ready to give the algorithm, which is a standard dynamic programming algorithm. Define a recurrence for $DP[s]$, which is the minimum number of TFs that can do a subset s of the jobs. For the base case we have $DP[\varnothing] = 0$. We arrange the rest of the problems by size, with our recurrence being

$$DP[s] = \min_{s' \subset s}(DP[s \setminus s'] + 1)$$

where s' loops over all subsets s' that a single TF can do. Once we calculate all of the DP values, we check if $DP[J] \leq k$ where J is the set of jobs is complete. We claim that

$$DP[J] \leq k \quad \iff \quad k \text{ TFs are sufficient to do the jobs in } J.$$

We prove this by induction. In the base case, it is clearly true since the number of TFs required to do an empty subset of the jobs is 0. Now suppose we had some unique subset $s \subset J$ of jobs. Assume that $DP[s']$ satisfies the claim for all $s' \subset s$. Since s is nonempty, at least 1 TF is required, but by the inductive hypothesis, $DP[s-t]$ is equal to the minimum number of TFs required to do $s-t$ for every $t \subset s$ of tasks that one TF can complete. Thus the minimum number of TFs required to do the tasks in s is $DP[s-t]+1$. The minimum of this expression is then guaranteed to be the minimun number of TFs required to do s of the tasks.

Let's now calculate the complexity of the algorithm. Our dynamic programming has a state for every unique subset of jobs, so there are $O(n^{\ell})$ states by the hint we proved. We also must iterate over at most every unique subset of tasks that a single TF can complete, which as we proved was $O(m/\min(T))^{\ell}$. We can calculate the value of $s - s'$ in $O(\ell)$ time, so the time taken to calculate the value of all dynamic programming states is $O((m/\min(T))^{\ell} + \ell)$ which becomes $O((m/min(T))^{\ell})$ if ℓ is assumed to be a constant. Putting everything together, the overall runtime is

$$O\left(\frac{nm}{\min(T)}\right)^{\ell}.$$

(c) Consider the reduction from $(1+\epsilon)$-TFHiring to Discrete-TFHiring defined as follows: We can assume $j_i \leq m$ for any job, otherwise we return false immediately. Then define a constant $c = m\epsilon^2$, and keep $n' = n$ and $k' = k$. For every job j_i in the original list, we set j_i' to j_i rounding up to the nearest multiple of c. We also set a list T as multiples of c between $m\epsilon$ and $m + c$. Lastly, set m' to $m(1+\epsilon)$. This is clearly a polynomial reduction of the problem.

To prove correctness, first we note that we can disregard the case when $j_i > m$ for some i, in this case the solution is false and we address this case. Next, let's consider the case when $m\epsilon \leq j_i \leq m$ for all j_i. Then clearly $j_i' \in T$ because $j_i' \leq m + c$ and $j_i' \geq m\epsilon$. Moreover, the number of multiples of c between $m\epsilon$ and $m + \epsilon$ is at most

$$\frac{m + c - m\epsilon}{c} + 1 = \frac{m - m\epsilon}{c} + 2.$$

Substituting $m\epsilon^2$ for c, this simplifies to $\frac{1}{\epsilon^2} - \frac{1}{\epsilon} + 2$. Assuming that $\epsilon \leq \frac{1}{2}$, this value is $\leq \frac{1}{\epsilon^2}$. Then by the definition of T, we have $|T| \leq \frac{1}{\epsilon^2}$ so we have a valid input for Discrete-TFHiring.

Next we show that this reduction gives the same solution to both problems. Suppose we had a solution to TFHiring which assigns job j_i to some TF g_i where $1 \leq g_i \leq k$. We claim that if we the job j'_i to the TF g_i, we get a valid solution to Discrete-TFHiring. Recall that $j'_i \leq j_i + c$ for all i. Now consider a TF g and lets look at the subset J_g of jobs that were assigned to g. This is a solution to TFHiring, so we know that $\sum j_i \leq m$ for all jobs $j_i \in J_g$ so we have

$$\sum j'_i \leq \sum (j_i + c) \leq m + c \cdot |J|.$$

Since every job takes at least $m\epsilon$ minutes, the maximum number of jobs a single TF can do is $m/m\epsilon = 1/\epsilon$. So $|J| \leq m/m\epsilon$ so $m + c \cdot |J| \leq m + c/\epsilon$. Again substituting $m\epsilon^2$ for c sets this equal to $m(1+\epsilon)$. So in conclusion, we see that for every TF, $\sum j'_i \leq m(1+\epsilon)$ and so this is a valid solution to Discrete-TFHiring.

Conversely, if there is no way to assign jobs in TFHiring such that no TF works more than $m(1+\epsilon)$ minutes per week, lets assume for the sake of contradiction that Discrete-TFHiring returns true. Then there is some allocation of task such that for every TF we have $\sum j'_i \leq m(1+\epsilon)$. However since $j'_i \geq j_i$ we get that $\sum j_i \leq m(1+\epsilon)$ so there is a way to assign jobs in the TFHiring, a contradiction to an assumption.