CSE 6140 / CX 4140

Computational Science & Engineering Algorithms

Homework 4

Please type all answers.

1. (15 points)

A store trying to analyze the behavior of its customers will often maintain a two-dimensional array A, where the rows correspond to its customers and the columns correspond to the products it sells. The entry $A[i, j]$ specifies the quantity of product j that has been purchased by customer i.

Here's a tiny example of such an array A.

	liquid detergent	**beer**	**diapers**	**cat litter**
Raj	0	6	0	3
Alanis	2	3	0	0
Chelsea	0	0	0	7

One thing that a store might want to do with this data is the following. Let us say that a subset S of the customers is *diverse* if no two of the of the customers in S have ever bought the same product (i.e., for each product, at most one of the customers in S has ever bought it). A diverse set of customers can be useful, for example, as a target pool for market research.

We can now define the Diverse Subset Problem as follows: Given an $m \times n$ array A as defined above, and a number $k \leq m$, is there a subset of at least k of customers that is *diverse*?

Show that Diverse Subset is NP-complete.

Hint: Independent Set is NP-complete.

The Diverse Subset problem is in NP because, given a subset S of customers, we can verify in polynomial time whether S is diverse. For each product, we check that no two customers in S have both purchased the same product. This verification can be done by examining each column in the matrix A, ensuring that at most one customer in S has a non-zero entry for each product. Since this check can be done in polynomial time, the Diverse Subset problem belongs to NP.

We can reduce the *Independent Set* problem, which is known to be NP-complete, to the Diverse Subset problem. Given a graph $G = (V, E)$ and an integer k, the Independent Set problem asks whether there exists an independent set of size k.

We transform this problem into an instance of the Diverse Subset problem:

- Create a customer for each vertex in G
- Create a product for each edge in G
- For each customer v and product e, set $A[v][e] = 1$ if edge e is incident to vertex v, otherwise set $A[v][e] = 0$
- Ask if there exists a diverse subset of k customers such that no two customers have bought the same product

A diverse subset of size k exists if and only if there is an independent set of size k in G.

Forward direction: If there is an independent set of size k, no two vertices in this set share an edge. Therefore, in the corresponding matrix, no two customers will have purchased the same product, forming a diverse subset.

Backward direction: If there is a diverse subset of size k, no two customers in the subset have bought the same product. This implies that the corresponding vertices in G share no edges, forming an independent set.

The reduction from the Independent Set problem to the Diverse Subset problem can be done in polynomial time, and Independent Set is NP-complete, thus, the Diverse Subset problem is also NP-complete.

2. (15 points)

Suppose you're helping to organize a summer sports camp, and the following problem comes up. The camp is supposed to have at least one counselor who's skilled at each of the n sports covered by the camp (baseball, volleyball, and so on). They have received job applications from m potential counselors. For each of the n sports, there is some subset of the m applicants qualified in that sport. The question is: For a given number $k < m$, is it possible to hire at most k of the counselors and have at least one counselor qualified in each of the n sports? We'll call this the *Efficient Recruiting Problem*.

Show that Efficient Recruiting is NP-complete.

Hint: Vertex Cover in NP-complete.

Given a subset S of customers, we can verify in polynomial time whether S is diverse. For each product, we check that no two customers in S have both purchased the same product. This verification can be done by examining each column in the matrix A, ensuring that at most one customer in S has a non-zero entry for each product. Since this check can be done in polynomial time, the Diverse Subset problem belongs to NP.

We can reduce the *Independent Set* problem, which is known to be NP-complete, to the Diverse Subset problem. In the *Independent Set* problem, given a graph $G = (V, E)$ and a number k, we ask whether there exists an independent set of size k.

We can transform this problem into an instance of the Diverse Subset problem:

- Create a customer for each vertex in G
- Create a product for each edge in G
- For each customer v and product e, set $A[v][e] = 1$ if edge e is incident to vertex v, otherwise set $A[v][e] = 0$
- Ask if there exists a diverse subset of k customers such that no two customers have bought the same product

A diverse subset of size k exists if and only if there is an independent set of size k in G.

- *Forward direction:* If there is an independent set of size k, no two vertices in this set share an edge. Therefore, in the corresponding matrix, no two customers will have purchased the same product, forming a diverse subset.
- *Backward direction:* If there is a diverse subset of size k, no two customers in the subset have bought the same product. This implies that the corresponding vertices in G share no edges, forming an independent set.

Therefore, since the reduction can be performed in polynomial time, and Independent Set is NP-complete, the Diverse Subset problem is also NP-complete.

3. (20 points)

The mapping of genomes involves a large array of difficult computational problems. At the most basic level, each of an organism's chromosomes can be viewed as an extremely long string (generally containing millions of symbols) over the four-letter alphabet $\{A, C, G, T\}$. One family of approaches to genome mapping is to generate a large number of short, overlapping snippets from a chromosome, and then to infer the full long string representing the chromosome from this set of overlapping substrings.

While we won't go into these string assembly problems in full detail, here's a simplified problem that suggests some of the computational difficulty one encounters in this area. Suppose we have a set $S = \{s_1, s_2, \ldots, s_n\}$ of short DNA strings over a q-letter alphabet; and each string s_i has length 2ℓ, for some number $\ell \geq 1$. We also have a library of additional strings $T = \{t_1, t_2, \ldots, t_m\}$ over the same alphabet; each of these also has length 2ℓ. In trying to assess whether the string s_b might come directly after the string s_a in the chromosome, we will look to see whether the library T contains a string t_k so that the first ℓ symbols in t_k are equal to the last ℓ symbols in s_a, and the last ℓ symbols in t_k are equal to the first ℓ symbols in s_b. If this is possible, we will say that t_k *corroborates* the pair (s_a, s_b). (In other words, t_k could be a snippet of DNA that straddled the region in which s_b directly followed s_a.)

Now we'd like to concatenate all the strings in S in some order, one after the other with no overlaps, so that each consecutive pair is corroborated by some string in the library T. That is, we'd like to order the strings in S as $s_{i_1}, s_{i_2}, \ldots, s_{i_n}$, where $i_1, i_2, \ldots, i_n$ is a permutation of $\{1, 2, \ldots, n\}$, so that for each $j = 1, 2, \ldots, n-1$, there is a string t_k that corroborates the pair $(s_{i_j}, s_{i_{j+1}})$. (The same string t_k can be used for more than one consecutive pair in the concatenation.) If this is possible, we will say that the set S has a *perfect assembly*.

Given sets S and T, the *Perfect Assembly Problem* asks: Does S have a perfect assembly with respect to T? Prove that Perfect Assembly is NP-complete.

Example. Suppose the alphabet is $\{A, C, G, T\}$, the set $S = \{AG, TC, TA\}$, and the set $T = \{AC, CA, GC, GT\}$ (so each string has length $2\ell = 2$). Then the answer to this instance of Perfect Assembly is yes: We can concatenate the three strings in S in the order $TCAGTA$ (so $s_{i_1} = s_2$, $s_{i_2} = s_1$, and $s_{i_3} = s_3$). In this order, the pair (s_{i_1}, s_{i_2}) is corroborated by the string CA in the library T, and the pair (s_{i_2}, s_{i_3}) is corroborated by the string GT in the library T.

Hint: Hamiltonian Path is NP-complete.

Given a permutation of strings $S = \{s_1, s_2, \ldots, s_n\}$ and a set T of corroborating strings, we can verify in polynomial time whether the given permutation forms a

perfect assembly. For each consecutive pair of strings $s_{i_j}, s_{i_{j+1}}$, we check if there exists a string $t_k \in T$ such that the last ℓ symbols of s_{i_j} match the first ℓ symbols of $s_{i_{j+1}}$. Since this verification can be done in polynomial time, the problem is in NP.

We will now reduce the *Hamiltonian Path* problem, which is known to be NP-complete, to the Perfect Assembly Problem.

The Hamiltonian Path problem asks whether, given a directed graph $G = (V, E)$ with n vertices, there exists a path that visits each vertex exactly once.

We transform this problem into an instance of the Perfect Assembly Problem:

- For each vertex $v_i \in V$, create a string $s_i \in S$
- For each directed edge $(v_i, v_j) \in E$, create a corroborating string $t_k \in T$ such that the last ℓ symbols of s_i match the first ℓ symbols of s_j
- We then ask whether there exists a permutation of S that forms a perfect assembly with respect to T, such that every consecutive pair $(s_{i_j}, s_{i_{j+1}})$ is corroborated by a string $t_k \in T$

A perfect assembly of the set S exists if and only if there is a Hamiltonian Path in G.

- *Forward direction:* If there exists a Hamiltonian Path in G, the corresponding permutation of strings in S forms a perfect assembly. The directed edges in the Hamiltonian Path ensure that for every consecutive pair $(s_{i_j}, s_{i_{j+1}})$, there exists a corroborating string $t_k \in T$.
- *Backward direction:* If there exists a perfect assembly of S, the corresponding permutation of strings represents a Hamiltonian Path in G, where each corroborating string $t_k \in T$ corresponds to a directed edge in G between consecutive vertices.

Therefore, since the reduction from Hamiltonian Path to the Perfect Assembly Problem can be performed in polynomial time, and Hamiltonian Path is NP-complete, the Perfect Assembly Problem is also NP-complete.

4. (20 points) Madison is in preschool and has learned to spell some simple words. She has a colorful set of refrigerator magnets featuring the letters of the alphabet (some number of copies of the letter A, some number of copies of the letter B, and so on), and the last time you saw her the two of you spent a while arranging the magnets to spell out words that she knows. The two of you tried to spell out words so as to use up all the magnets in the full set –that is, picking words that she knows how to spell, so that once they were all spelled out, each magnet was participating in the spelling of exactly one of the words. (Multiple copies of words are okay here; so for example, if the set of refrigerator magnets includes two copies each of C, A, and T, it would be okay to spell out CAT twice.)

This turned out to be pretty difficult, and it was only later that you realized a plausible reason for this. Suppose we consider a general version of the problem of Using Up All the Refrigerator Magnets, where we replace the English alphabet by an arbitrary collection of symbols, and we model Madison's vocabulary as an arbitrary set of strings over this collection of symbols. The goal is the same as in the previous paragraph.

Prove that the problem of Using Up All the Refrigerator Magnets is NP-complete.

Hint: 3D matching is NP-complete.

Given a collection of magnets and a set of words that Madison knows, we can verify in polynomial time whether the magnets can be arranged to spell out some combination of the words, using up all the magnets exactly. For each letter in the magnets and each word spelled, we ensure that the count of letters used matches the count of magnets available. This verification can be done in polynomial time, so the problem belongs to NP.

We can now reduce the *3D Matching* problem, which is known to be NP-complete, to the Using Up All the Refrigerator Magnets problem.

The 3D Matching problem asks whether, given three disjoint sets X, Y, and Z, and a collection of 3-tuples $M \subseteq X \times Y \times Z$, there exists a matching $M' \subseteq M$ such that every element of X, Y, and Z is included in exactly one tuple of M'.

We can transform this problem into an instance of the Using Up All the Refrigerator Magnets problem:

- Create magnets corresponding to the elements of X, Y, and Z, where each element has one magnet
- Create a word for each 3-tuple in M, where each word consists of the three letters corresponding to the elements in the tuple
- Ask if we can spell out words using the magnets such that all the magnets are used up exactly once

The magnets can be used to spell out words if and only if there is a perfect matching

of the set $X \times Y \times Z$.

- *Forward direction:* If there is a perfect matching of $X \times Y \times Z$, then we can use the words corresponding to the tuples in the matching to spell out all the magnets exactly. Each word uses up exactly three magnets corresponding to the elements in the tuple, and since the matching contains all elements, all magnets will be used up exactly once.
- *Backward direction:* If we can use up all the magnets to spell out words, this corresponds to selecting a matching from M such that each element of X, Y, and Z is used exactly once. This forms a perfect matching.

Therefore, since the reduction from the 3D Matching problem to the Using Up All the Refrigerator Magnets problem can be performed in polynomial time, and 3D Matching is NP-complete, the Using Up All the Refrigerator Magnets problem is also NP-complete.

5. (15 points)

A convoy of ships arrives at a port and delivers a total of n containers, each containing a different kind of hazardous material. Waiting near the port is a set of m trucks, each of which can hold up to k containers. Any container can be placed in any truck; however, there are certain pairs of containers that cannot be placed together in the same truck. The chemicals they contain may react explosively if brought into contact.

The *Truck Loading Problem* is. Is there a way to load all n containers into the m trucks so that no truck is overloaded, and no two containers are placed in the same truck when they are not supposed to be?

Show that Truck Loading is NP-complete.

Hint: 3-Coloring is NP-complete.

Given a loading configuration of trucks and containers, we can verify in polynomial time whether all the containers are loaded in such a way that no truck is overloaded (each holds at most k containers) and no two containers that should not be placed together are in the same truck. This verification can be done in polynomial time, so the problem belongs to NP.

We can now reduce the *3-Coloring* problem, which is known to be NP-complete, to the Truck Loading problem.

The 3-Coloring problem asks whether, given a graph $G = (V, E)$, it is possible to color the vertices of G with three colors such that no two adjacent vertices share the same color.

We transform this problem into an instance of the Truck Loading problem:

- Create a container for each vertex in the set V
- Create a truck for each of the three colors, where each truck can hold up to $k = \lceil n/3 \rceil$ containers
- For each edge $(u, v) \in E$, specify that the containers corresponding to u and v cannot be placed in the same truck
- Ask if we can load all the containers into the trucks such that no truck holds more than k containers and no containers that correspond to adjacent vertices are placed in the same truck

The containers can be loaded into the trucks if and only if there is a valid 3-coloring of the graph.

- *Forward direction:* If there is a valid 3-coloring of the graph, we can assign each vertex to a truck corresponding to its color, ensuring that no adjacent vertices

(i.e. no incompatible containers) are placed in the same truck. Additionally, since each color class forms an independent set of vertices, no truck will hold more than $\lceil n/3 \rceil$ containers.

- *Backward direction:* If we can load all the containers into the trucks such that no containers that correspond to adjacent vertices are placed together, this corresponds to a valid 3-coloring of the graph where each vertex is assigned the color of the truck in which its container is placed.

Therefore, since the reduction from the 3-Coloring problem to the Truck Loading problem can be performed in polynomial time, and 3-Coloring is NP-complete, the Truck Loading problem is also NP-complete.

6. (15 points)

One thing that's not always apparent when thinking about traditional "continuous math" problems is the way discrete, combinatorial issues of the kind we're studying here can creep into what look like standard calculus questions.

Consider, for example, the traditional problem of minimizing a one-variable function like $f(x) = 3 + x - 3x^2$ over an interval like $x \in [0, 1]$. The derivative has a zero at $x = 1/6$, but this in fact is a maximum of the function, not a minimum; to get the minimum, one has to heed the standard warning to check the values on the boundary of the interval as well. (The minimum is in fact achieved on the boundary, at $x = 1$.)

Checking the boundary isn't such a problem when you have a function in one variable; but suppose we're now dealing with the problem of minimizing a function in n variables $x_1, x_2, \ldots, x_n$ over the unit cube, where each of $x_1, x_2, \ldots, x_n \in [0, 1]$. The minimum may be achieved on the interior of the cube, but it may be achieved on the boundary; and this latter prospect is rather daunting, since the boundary consists of 2^n "corners" (where each x_i is equal to either 0 or 1) as well as various pieces of other dimensions. Calculus books tend to get suspiciously vague around here, when trying to describe how to handle multivariable minimization problems in the face of this complexity.

It turns out there's a reason for this: Minimizing an n-variable function over the unit cube in n dimensions is as hard as an NP-complete problem. To make this concrete, let's consider the special case of polynomials with integer coefficients over n variables $x_1, x_2, \ldots, x_n$. To review some terminology, we say a *monomial* is a product of a real-number coefficient c and each variable x_i raised to some nonnegative integer power a_i; we can write this as $cx_1^{a_1}x_2^{a_2} \cdots x_n^{a_n}$. (For example, $2x_1^2x_2x_3^4$ is a monomial.) A *polynomial* is then a sum of a finite set of monomials. (For example, $2x_1^2x_2x_3^4 + x_1x_3 - 6x_2^2x_3^2$ is a polynomial.)

We define the *Multivariable Polynomial Minimization Problem* as follows: Given a polynomial in n variables with integer coefficients, and given an integer bound B, is there a choice of real numbers $x_1, x_2, \ldots, x_n \in [0, 1]$ that causes the polynomial to achieve a value that is $\leq B$?

Prove that *Multivariable Polynomial Minimization* is NP-complete.

Hint: 3-SAT is NP-complete.

Given a polynomial in n variables and an integer bound B, we can verify in polynomial time whether the polynomial achieves a value less than or equal to B for some choice of real numbers $x_1, x_2, \ldots, x_n \in [0, 1]$. This verification involves evaluating

the polynomial at a given assignment of the variables and checking if the resulting value is less than or equal to B. Since this check can be done in polynomial time, the problem belongs to NP.

We can now reduce the *3-SAT* problem, which is known to be NP-complete, to the Multivariable Polynomial Minimization Problem.

The 3-SAT problem asks whether, given a Boolean formula in conjunctive normal form with three literals per clause, there exists an assignment to the variables such that the formula evaluates to true.

We transform this problem into an instance of the Multivariable Polynomial Minimization Problem:

- For each Boolean variable in the 3-SAT instance, create a corresponding variable $x_i \in [0, 1]$
- For each clause in the 3-SAT instance, create a polynomial that evaluates to 0 if the clause is satisfied and a positive value otherwise
- Define the overall polynomial as the sum of the polynomials for each clause, and set the bound $B = 0$
- Ask whether there is an assignment to the variables such that the polynomial evaluates to a value less than or equal to $B = 0$

The polynomial can be minimized to a value less than or equal to $B = 0$ if and only if there is a satisfying assignment for the 3-SAT formula.

- *Forward direction:* If there is a satisfying assignment for the 3-SAT formula, the corresponding assignment of values to the variables in the polynomial will cause each clause polynomial to evaluate to 0, and hence the sum of the clause polynomials will be 0, which is less than or equal to $B = 0$.
- *Backward direction:* If the polynomial can be minimized to a value of 0, this means that each clause polynomial evaluates to 0, which implies that the corresponding assignment of Boolean variables satisfies the 3-SAT formula.

Therefore, since the reduction from the 3-SAT problem to the Multivariable Polynomial Minimization Problem can be performed in polynomial time, and 3-SAT is NP-complete, the Multivariable Polynomial Minimization Problem is also NP-complete.