HW4 Solutions

1.

The problem is in $\mathcal{NP}$ because we can exhibit a set of k customers, and in polynomial time is can be checked that no two bought any product in common.

We now show that *Independent Set* $\leq_P$ *Diverse Subset*. Given a graph G and a number k, we construct a customer for each node of G, and a product for each edge of G. We then build an array that says customer v bought product e if edge e is incident to node v. Finally, we ask whether this array has a diverse subset of size k.

We claim that this holds if and only if G has an independent set of size k. If there is a diverse subset of size k, then the corresponding set of nodes has the property that no two are incident to the same edge — so it is an independent set of size k. Conversely, if there is an independent set of size k, then the corresponding set of customers has the property that no two bought the same product, so it is diverse.

2.

The problem is in NP since, given a set of k counselors, we can check that they cover all the sports.

Suppose we had such an algorithm $\mathcal{A}$; here is how we would solve an instance of *Vertex Cover*. Given a graph $G = (V, E)$ and an integer k, we would define a sport S_e for each edge e, and a counselor C_v for each vertex v. C_v is qualified in sport S_e if and only if e has an endpoint equal to v.

Now, if there are k counselors that, together, are qualified in all sports, the corresponding vertices in G have the property that each edge has an end in at least one of them; so they define a vertex cover of size k. Conversely, if there is a vertex cover of size k, then this set of counselors has the property that each sport is contained in the list of qualifications of at least one of them.

Thus, G has a vertex cover of size at most k if and only if the instance of *Efficient Recruiting* that we create can be solved with at most k counselors. Moreover, the instance of *Efficient Recruiting* has size polynomial in the size of G. Thus, if we could determine the answer to the *Efficient Recruiting* instance in polynomial time, we could also solve the instance of *Vertex Cover* in polynomial time.

3.

To see that the *Perfect Assembly* problem is in NP we use the sequence forming the perfect assembly as the certificate. It is easy to check that a given sequence forms a perfect assembly.

To show that the problem is NP-complete, we show that *Hamiltonian Path* $\leq_P$ *Perfect Assembly.* Given an arbitrary instance of the *Hamiltonian Path* problem $G = (V, E)$, we use $\ell = 1$ and use 2 letters v_{in} and v_{out} for each node v in G. The set S of required sequences will be $S = \{v_{in}v_{out}$ for all $v \in V\}$, so a perfect assembly corresponds to ordering the nodes of the graph G. We set T of possible corroborating strings will correspond to edges of E. Let $T = \{v_{out}w_{in}$ for edges $(v, w) \in E\}$. We claim that this instance of *Perfect Assembly* is equivalent to the original *Hamiltonian Path* problem. To prove this assume first that $G = (V, E)$ has a Hamiltonian Path. Order the pairs in S in the order the path traverses the corresponding nodes. This forms a perfect assembly as the edges (v, w) of the path provide the necessary corroborating sequences in T. To see the other direction, assume there is a perfect assembly of S. This ordering of S corresponds to an ordering of the nodes on G, and the ordering is a Hamiltonian path in G, as whenever a node v is followed by a node w in the ordering, by the definition of perfect assembly the sequence $v_{in}v_{out}w_{in}w_{out}$ must be corroborated by a sequence $v_{out}w_{in}$ in T, and hence (v, w) must be an edge of G.

4. *Magnets* is in NP.

Certificate: A multiple set of words.

Certifier: Counts the number of each kind of magnets used in these words, and verifies whether that is equal to the given number of that kind of magnets. Done in polynomial time.

Now we will show that *Magnets* is NP-complete by reducing from *3D Matching*. We have an instance of *3D Matching*, which consists of three sets *X, Y, Z*, such that |*X*| = |*Y*| = |*Z*| = *n*, and a set of tuples *M*. We want to find *n* tuples from *M*, s.t. each element is covered by exactly one tuple. Create an instance of *Magnets* as follows: each element in *X, Y,* or *Z* becomes a magnet with a unique letter (so our alphabet will have *3n* letters), and every tuple (x_i, y_j, z_k) becomes a word that Madison knows. Solving this instance of *Magnets* will solve the instance of *3D Matching*.

We must now show that there is a perfect matching in the *3D Matching* problem if and only if all the magnets can be used up in the *Magnets* problem. If all the magnets are used up, we must have got exactly *n* words, since each word has 3 letters, and there are a total of *3n* letters, and therefore we have the desired *n* tuples. If there is a perfect matching in *3D Matching*, then those words that are corresponding to the tuples in the matching will exactly use up all the magnets without overlapping.

5.

We show how to reduce *3-Coloring* to this problem. Given a graph G on n nodes, we define a cannister i for each node v_i. We have three trucks, each of capacity $k = n$, and we say that two cannisters cannot go in the same truck whenever there is an edge between the corresponding nodes in G.

Now, if there is a 3-coloring of G, then we can place all the cannisters corresponding to nodes assigned the same coloring in a single truck. Conversely, if there is a way to place all the cannisters in the three trucks, then we can use the truck assignments as colors; this gives a 3-coloring of the graph.

6. *Multivariate Polynomial Minimization* is in NP.

Certificate: Assignments of the variables in the polynomial.

Certifier: Evaluating the polynomial and checking if the answer is within the bound B. (Done in polynomial time – polynomial in the number of variables and monomials).

For the reduction, we need to take an arbitrary instance of *3-SAT* and construct a particular instance of *Multivariate Polynomial Minimization*.

Consider an instance of *3-SAT.* Define a real variable y_i in the polynomial for each Boolean variable x_i in the *3-SAT* instance. Each clause becomes a product of three terms in the variables {y_i}.

We will represent a clause like with the product . That is, we represent in the clause with in the product, and in the clause by in the product. If we are given an assignment like (note that this is a satisfying assignment), we'll set , and the product will evaluate to 0.

Each clause in the *3-SAT* instance will be represented by a product. Note that by the distributive law of multiplication, each product can be written as a sum of at most eight monomials. For example can be expanded into a sum of four monomials.

From the *3-SAT* instance, we will construct a polynomial as follows. We first represent each clause by a product as described above. Then we take the sum of all these products. The products can be expanded into monomials to express the full polynomial. This is our instance of the *Multivariate Polynomial Minimization* problem. We will define our bound *B* to be zero.

For thinking about the reduction, it is useful to think about the polynomial in its form before we expanded the products (that is, before we applied the distributive law). There is one product for each clause. In order for the polynomial's value to be , each of these products must be 0; and the only way for this to happen is for one of the terms in the corresponding clause to be set so that it satisfies the clause. The polynomial can achieve a value of if and only if the original *3-SAT* instance was satisfiable.