2 Some Sums

(a) Suppose we are given some A with target sum t. Let s be the sum of all elements in A. If $s - 2t \geq 0$, generate a new set $A' = A \cup \{s - 2t\}$. If A' can be partitioned, then there is a subset of A that sums to t.

We know that the two sets in our partition must each sum to $s - t$ since the sum of all elements will be $2s - 2t$. One of these sets, must contain the element $s - 2t$. Thus the remaining elements in this set sum to t.

If $s - 2t \leq 0$, generate a new set $A' = A \cup \{2t - s\}$. If A' can be partitioned, then there is a subset of A that sums to t.

We know that the two sets in our partition must each sum to t since the sum of all elements will be $2t$. The set that does not contain $\{2t - s\}$ will be our solution to subset sum.

This reduction also clearly operates in $\mathcal{O}(n)$, as we simply need to generate a new set with a single additional element (whose value is determined by summing all the elements of A).

(b) Suppose we are given some set A with target sum t. For each element k of the set, create an item with weight k and value k. Let $V = t$ and $W = t$. We know Knapsack will determine if there is a combination of items with sum of weights $\leq t$ and values $\geq t$. Because the weights and values are the same, we know (Sum of chosen weights) = (Sum of chosen values) = t. And since each weight/value pair is exactly the value of one of the original elements of A, we know that there will be a solution to our Knapsack problem iff there is one for our subset sum problem. This solution runs in linear time, as we need constant work for each element of A to generate our new input.

3 Max k-XOR

(a) We create a variable for each vertex i, and a clause x_i XOR x_j for each edge (i, j). We choose $c = r$.

For any assignment, consider the cut such that S contains all vertices i for which xi is true in this assignment. For each edge (i, j) crossing this cut, its corresponding clause is true because exactly one of x_i, x_j is true. For each edge not crossing this cut, its corresponding clause is false because either both x_i, x_j are true or neither is true. This proves correctness of the reduction.

(b) The reduction is to add a new variable y and add y to every clause in the Max 3-XOR instance and choose the same value of r to get a Max 4-XOR instance.

Given an assignment that satisfies r clauses in the Max 3-XOR instance, the same assignment plus y set to false satisfies r clauses in the Max 4-XOR instance. Given an assignment that satisfies r clauses in the Max 4-XOR instance, if y is false, then the same assignment minus y satisfies r clauses in the Max 3-XOR instance. Otherwise, y is true, and the same assignment minus y and negating all other variables satisfies r clauses in the Max 3-XOR instance.

To see this, consider any clause that was true in Max 4-XOR. Deleting y takes it from having an odd number of true variables to an even number. Then, since 3 is odd, negating all variables brings it back to having an odd number of true variables. Similarly, any clause that was false in the Max 4-XOR instance has an even number of true variables. Deleting y causes it to have an odd number of true variables, and then negating all other variables brings it back to even.

4 Dominating Set

Proof that Minimum Dominating Set is in NP.

Given a possible solution, we can check that it is a solution by iterating through the vertices not in the solution subset D and checking if it has a neighbor in D by iterating through the adjacent edges. This would take at most E time because you would at most check every edge twice. We should also check that the number of vertices in D is less than k, which would take at most V as k should be less than E. So, we can verify in $\mathcal{O}(E)$ time which is polynomial.

Proof that Minimum Dominating Set is NP-Hard.

Reduction from Vertex Cover to Dominating Set

Minimum Vertex Cover is to find a vertex cover (a subset of the vertices) which is of $size \leq k$ where all edges have an endpoint in the vertex cover.

Suppose $G(V, E)$ is an instance of vertex cover and we are trying to find a vertex cover of $size \leq k$. For every edge (u, v), we will add a new vertex c and two edges (c, u), and (c, v). We will pass in the same k to the dominating set. This is a polynomial reduction because we are adding $|E|$ vertices and $2|E|$ edges.

Proof of Correctness of Reduction

1. If minimum vertex cover has a solution, then minimum dominating set that corresponds to it has a solution.

Suppose we have some minimum vertex cover of size k. By definition of vertex cover, every edge has either one or both endpoints in the vertex cover. Thus if we say that the vertex cover is a dominating set, every original vertex must either be in the dominating set or adjacent to it. Now we have to figure out whether the vertices we added for every edge are covered. Since every edge has to have one or both endpoint in the vertex cover, the added vertices must be adjacent to at least one vertex in the vertex cover. Thus the vertex cover maps directly to a dominating set in the transformed problem.

2. If minimum dominating set in the transformed format has a solution, then the corresponding minimum vertex cover has a solution.

Suppose we have some minimum dominating set of size k of the transformed format. Vertices in the dominating set either must come from the original vertices or the vertices we added. If they don't come from the vertices we added in the transformation, then from the logic stated in the previous direction, the dominating set corresponds directly to the vertex cover. If some vertices come from the transformation, then we can substitute the vertex for either of the endpoints of the edge it corresponds to without changing the size of the dominating set. Thus, we can come up with a dominating set of size k that only uses vertices from our original problem, which will directly match to a minimum vertex cover of size k in the original problem.

Alternative Proof that Minimum Dominating Set is NP-Hard.

Reduction from Set Cover to Dominating Set

Minimum Set Cover is to find a set cover (a subset of all the sets) which is of $size \leq k$ where all elements are covered by at least one set of the set cover.

Suppose (S, U) is an instance of set cover where U denotes the set of all distinct elements and S is a set of subsets S_i of U. We will construct a graph $G = (V, E)$ as follows. For each element u in U construct a vertex; we will call these "element vertices". For each possible S_i construct a vertex; we will call these "set vertices". Connect each vertex S_i to all u in S_i. Notice that if we were to run dominating set on the graph right now, we would be able to cover all the element vertices with any valid set cover, but we would have to pick every single set vertex in order to ensure that all set vertices are covered. To rectify this, connect every set vertex to every other set vertex, forming a clique. This ensures that we can cover all the set vertices by picking just one. This way we really only need to worry about covering the element vertices.

Proof of Correctness of Reduction

1. If minimum set cover has a solution, then minimum dominating set that corresponds to it has a solution.

Suppose we have some minimum set cover of size k. This will correspond to a dominating set of size k as well. For each set in our minimum set cover, pick the corresponding set vertex. It follows directly from the construction of the graph and the definition of a set cover that all set and element vertices are covered.

2. If minimum dominating set in the transformed format has a solution, then the corresponding minimum set cover has a solution.

Suppose we have some dominating set D of size k. We can find a set cover of $size \leq k$. To do this, we will construct a new dominating set D' that contains only set vertices. Include every set vertex in D in D'. For each vertex in our dominating set that is an element vertex, pick any random connecting set vertex and add it to D'. Observe that $|D'| \leq |D|$. Thus if there is a dominating set in G of $size \leq k$, there must be a set cover of $size \leq k$.

Since this problem is in NP and is NP-Hard, it must be NP-Complete.

5 Orthogonal Vectors (Optional)

We use an $\mathcal{O}(2^{n/2}m)$-time reduction from 3-SAT to orthogonal vectors. We split the variables into two groups of size $n/2, V_1, V_2$. For each group, we enumerate all $2^{n/2}$ possible assignments of these variables. For each assignment x of the variables in V_1, let v_x be the vector where the ith entry is 0 if the ith clause is satisfied by one of the variables in this assignment, and 1 otherwise. We ignore the variables in the clause that are in V_2. For example, if clause i only contains variables in V_2, then $v_x(i)$ for all x. Let A be the $2^{n/2}$ vectors produced this way.

We construct B containing $2^{n/2}$ vectors in a similar manner, except using V_2 instead of V_1.

We claim that the 3-SAT instance is satisfiable if and only if there is an orthogonal vector pair in $A \times B$. Given this claim, we can solve 3-SAT by making the orthogonal vectors instance in $\mathcal{O}(2^{n/2}m)$ time, and then solving the instance in $\mathcal{O}((2^{n/2})^c m) = \mathcal{O}(2^{cn/2}m)$ time. Suppose there is a satisfying assignment to 3-SAT. Let x_1 be the assignment of variables in V_1, and x_2 be the assignment of variables in V_2. Let v_1, v_2 be the vectors in A, B corresponding to x_1, x_2. Since every clause is satisfied, one of $v_1(i)$ and $v_2(i)$ must be 0 for every i, and so $v_1 \cdot v_2 = 0$. So there is also a pair of orthogonal vectors in the orthogonal vectors instance. Suppose there is a pair of orthogonal vectors v_1, v_2 in the orthogonal vectors instance. Then for every i, either $v_1(i)$ or $v_2(i)$ is 0. In turn, for the corresponding assignment of variables in V_1, V_2, the combination of these assignments must satisfy every clause. In turn, the combination of these assignments is a satisfying assignment for 3-SAT.

Comment: It is widely believed that SAT has no $\mathcal{O}(2^{0.999n}m)$-time algorithm - this is called the Strong Exponential Time Hypothesis (SETH). So it is also widely believed that orthogonal vectors has no $\mathcal{O}(n^{1.99}m)$-time algorithm, since otherwise SETH would be violated. It turns out that we can reduce orthogonal vectors to string problems such as edit distance and longest common subsequence, and so if we belive SETH then we also believe those problems also don't have $\mathcal{O}(n^{1.99})$-time algorithms. The field of research studying reductions between problems with polynomial-time algorithms such as these is known as fine-grained complexity, and orthogonal vectors is one of the central problems in this field.

6 Understanding and Starting Your Project!

(a) We try to simplify this formula:

$$\begin{aligned} C_{\vec{p}} &= \exp\left(70\sqrt{\|\vec{b}\|_2^2 - b_i^2 - b_j^2 + \left(b_i - \frac{1}{|V|}\right)^2 + \left(b_j + \frac{1}{|V|}\right)^2}\right) \\ &= \exp\left(70\sqrt{\|\vec{b}\|_2^2 + \frac{2}{|V|^2} + \frac{2\left(b_j - b_i\right)}{|V|}}\right). \end{aligned}$$

While we have $\vec{b}$ and $\|\vec{b}\|_2$, b_i and b_j can be attained in $\mathcal{O}(1)$. Thus this can be computed in $\mathcal{O}(1)$ time.

(b) Say that we replace $\pi_2(v)$ from $\pi_1(v)$, therefore some team-ids are changed from i to j for some $v \in V$ and $i \neq j$. We define $u \in V$ as neighbours of any $v \in \mathit{team}(i) \cup \mathit{team}(j)$, D as a set of edges between v and u, and then update C_w as follows:

$$C_w = \sum_{\substack{(u,v)\in E \\ \pi_1(u)=\pi_1(v)}} w_{uv} - \sum_{\substack{(v,u)\in D \\ \pi_2(u)=i}} w_{vu} + \sum_{\substack{(v,u)\in D \\ \pi_2(u)=j}} w_{vu}.$$

Since we find u at most $|V|-1$ vertices for a certain vertex v, $|D|$ can be at most $|V|-1$. The total time spent is $\mathcal{O}(|V|)$.

(c) Suppose that we have known D's transformation from $\pi_1(v)$ to $\pi_2(v)$, then we can iterate these steps for every $d \in D$ as follows:

- change d's team-id from $\pi_1(d)$ to $\pi_2(d)$, then update $\vec{b}$ and $\|\vec{b}\|_2$
- update C_w using the algorithm in part (b), which takes $\mathcal{O}(|V|)$ time.

By the end of the iteration, compute $C_{\vec{p}}$ using updated $\vec{b}$ and $\|\vec{b}\|_2$, then we can calculate C_{π_2} for take $\mathcal{O}(|D||V|)$ temporal expenditure.

(d) we could making optimal partition in $k = 4$, in the circumstance where $\vec{b} = \frac{\vec{p}}{|V|} - \frac{\vec{1}}{k} = \vec{0}$: