Subset-Sum and Intro to Network Flow

1 Subset-Sum

SUBSET-SUM (SS): Given $a_1, a_2, \ldots, a_n \in \mathbb{N}$ and a target integer b, is there a set $S \subseteq \{1, 2, \ldots n\}$ with $\sum_{i \in S} a_i = b$?

Theorem 1. *SS is **NP**-complete.*

Proof. A two-part proof: (1) SS $\in$ **NP**, (2) SS is **NP**-hard.

(1). SS $\in$ **NP**

Certificate: A proposed set S.

Certifier: A polytime algorithm that checks $\sum_{i \in S} a_i = b$ and that $S \subseteq \{1, 2, \ldots n\}$.

(2). SS is **NP**-hard

We reduce from a known **NP**-hard problem (X3C) to SS.

Theorem 2. *X3C $\leq p$ SS.*

Proof. The input to the reduction is the set X with $|X| = 3m$ and $T_1, T_2, \ldots, T_n \subseteq X$ with $|T_i| = 3$. The output is the natural numbers $a_1, a_2, \ldots, a_k$ and b.

Observe that the sets T_j are representable as bitstrings. We put X in an arbitrary order $X = \{X_1, X_2, \ldots, X_{3m}\}$; then T_j is a $3m$-bit number, where there are three ones in the positions of X_i which are contained in T_j and zeroes everywhere else.

Let a_j be the number corresponding to the bitstring representation of T_j. If $T_i \cap T_j = \emptyset$, then $a_i + a_j$ has six ones exactly in the positions $T_i \cup T_j$. However, when $T_i \cap T_j \neq \emptyset$, there is carryover between the bits, which invalidates our mapping from sets to numbers. So, we read the bitstrings with a large enough base such that carryover can never happen (i.e., base $n + 1$).

Thus, we set a_j to the bitstring describing T_j read in base $n + 1$, $b = 11 \ldots 1_{n+1} = \sum_{i=0}^{3m-1} (n + 1)^i = \frac{1}{n}(n+1)^{3m} - 1$, and $k = n$. This reduction produces exponentially large numbers in b, but only polynomially many bits, so it runs in polynomial time. It satisfies:

(1). If X3C has a solution S, then SS also has a solution.

(2). If SS has a solution S with $\sum_{j \in S} a_j = 11 \ldots 1_{n+1}$, then X3C also has a solution. $\square$

Since we proved that SS is both in **NP** and **NP**-hard, SS must be **NP**-complete. In an earlier lecture, we saw SS $\leq p$ KNAPSACK, so we also know that KNAPSACK is **NP**-complete. $\square$

2 Partition

2.1 Partition

PARTITION (PR): Given $a_1, a_2, \ldots, a_n \in \mathbb{N}$, is there a set $S \subseteq \{1, 2, \ldots n\}$ with $\sum_{i \in S} a_i = \frac{1}{2} \sum_j a_j$?

Theorem 3. *PR is **NP**-complete.*

Proof. A two-part proof: (1) PR $\in$ **NP**, (2) PR is **NP**-hard.

(1). PR $\in$ **NP**

Certificate: A proposed set S.

Certifier: A polytime algorithm which checks that all elements exist and $\sum_{i \in S} a_i = \frac{1}{2} \sum_j a_j$.

(2). PR is **NP**-hard

We reduce from a known **NP**-hard problem (SS) to PR.

Theorem 4. *SS $\leq p$ PR*

Proof. The input to the reduction is $a_1, a_2, \ldots, a_n \in \mathbb{N}$ and b. The output is $a'_1, a'_2, \ldots, a'_m$. Without loss of generality, we say that $b \leq \frac{1}{2} \sum_i a_i$.

Let $m = n + 2$ and $a'_i = a_i$ for all $i \leq n$. With the remaining two a'_i elements, we will construct two large elements such that they cannot be placed in the same set, and they balance the sets to equivalent sizes. Let $W = \sum_i a_i$. Then $a'_{n+1} = 2W + b$, and $a'_{n+2} = 3W - b$.

This reduction is polytime and satisfies:

(1). If there exists an S such that $\sum_{i \in S} a_i = b$, then there exists a set S' such that $\sum_{i \in S} a'_i = \frac{1}{2} \sum_j a'_j$.

Add a'_{n+2} to S. Then, $\sum_{i \in S} a'_i + a'_{n+2} = b + 3W - b = 3W = \frac{1}{2} \sum_j a_j$.

(2). If there exists a set S' such that $\sum_{i \in S} a'_i = \frac{1}{2} \sum_j a'_j$, then there exists an S such that $\sum_{i \in S} a_i = b$.

We know that S' and $\bar{S}'$ contain exactly one of a'_{n+1} and a'_{n+2}. Say that $a'_{n+2} \in S'$; then $\sum_{i \in S, i \neq n+2} a'_i = 3W - (3W - b) = b$, so S' without a'_{n+2} solves SS. $\square$

Since we proved that PR is both in **NP** and **NP**-hard, PR must be **NP**-complete. $\square$

2.2 An Incorrect Reduction to Graph Partition

GRAPH PARTITION (GPR): Given a graph $\mathcal{G}$, is it possible to divide V into $S, \bar{S}$ such that $|S| = \frac{n}{2}$ and there are no edges between $(S, \bar{S})$?

Incorrect method to show that PR $\leq p$ GPR: Given $a_1, a_2, \ldots, a_m$, then for each a_i, construct a clique of a_i nodes. Then, there will always be such an equal, and because every component is a clique, there will be no edges in the cut. While the logic here is correct, the algorithm is not polytime. Instead, it is pseudo-polynomial, because building the m cliques is equivalent to writing every a_i in unary.

3 Network Flow

3.1 Definition of flow

Given a network $\mathcal{G} = (V, E)$ of links (pipes, streets, etc). Each link has a capacity $c_e \geq 0$ of how much flow it can transport per unit time at steady state. Find the maximum amount that can be transported from a source s to a sink t in steady state per unit time. For convenience of definiton, we assume no incoming edges to s, or outgoing edges from t.

A **flow** F assigns an amount of flow f_e to each edge e. The requirements of a flow are:

(1). Non-negative: $f_e \geq 0 \ \forall e$

(2). Capacity constraint: $f_e \leq c_e \ \forall e$

(3). Conservation constraint: $\forall v \neq s, t : \sum_{e \text{ into } v} f_e = \sum_{e \text{ out of } v} f_e$. "What goes in must come out."

The **flow value** $v(F) = \sum_{e \text{ out of } s} f_e$. Equivalently, $v(F) = \sum_{e \text{ into } t} f_e$.

3.2 Max flow

MAX-FLOW: Given $\mathcal{G}$, cost vector $\vec{c}$, and $s, t \in V$, find F^* from s to t maximizing $v(F)$. Note: F^* exists by a compactness and continuity argument.

Theorem 5. *There exists a polynomial time algorithm to find F^*; further, if all $c_e \in \mathbb{N}$, then all $f_e \in \mathbb{N}$.*

3.3 Min cut

MIN-CUT: Given $\mathcal{G}$, cost vector $\vec{c}$, and $s, t \in V$, find a cut $(S, \bar{S})$ of V with $s \in S$, $t \in \bar{S}$ that minimizes $C(S, \bar{S}) = \sum_{e=(u,v) \text{ across } (S,\bar{S})} c_e$. In other words, we are looking for the biggest bottleneck across a cut.

Theorem 6. *A minimum (s, t) cut with respect to c_e can be found in polynomial time.*

Theorem 7. *The cost of the minimum (s, t) cut with respect to c_e is equal to the maximum (s, t) flow with capacities c_e.*

Min cut is technically an upper bound to max flow, but next class we will prove that they elegantly equalize.