18.1 Review

18.1.1 Polynomial-Time Reduction

Definition 18.1 \( \pi_2 \) is polynomial-time reducible to \( \pi_1 \), \( (\pi_2 \leq_p \pi_1) \), if and only if there exists a polynomial-time algorithm \( f \) that transforms any instance \( x \) of \( \pi_2 \) to an instance \( f(x) \) of \( \pi_1 \) such that,

- \( x \) is a YES instance of \( \pi_2 \) \( \iff \) \( f(x) \) is a YES instance of \( \pi_1 \), and
- \( x \) is a NO instance of \( \pi_2 \) \( \iff \) \( f(x) \) is a NO instance of \( \pi_1 \).

![Diagram of polynomial-time reduction from \( \pi_2 \) to \( \pi_1 \).](image)

Figure 18.1: Polynomial time reduction from \( \pi_2 \) to \( \pi_1 \).

Lemma 18.2 If \( \pi_2 \leq_p \pi_1 \) and if \( \pi_1 \in \mathbf{P} \), then \( \pi_2 \in \mathbf{P} \).

![Diagram of polynomial-time reduction algorithm for \( \pi_2 \).](image)

Figure 18.2: Polynomial-time reduction algorithm for \( \pi_2 \).

Let function \( f \) be a polynomial-time reduction algorithm that computes \( f(x) \) for each input instance \( x \) of \( \pi_2 \). For a given input \( x \in \pi_2 \) the function \( f \) transforms \( x \) into \( f(x) \) and this can be used as input to \( \pi_1 \) which
outputs YES or NO answers for the original problem $\pi_2$. The transformation takes polynomial-time and so does solving problem $\pi_1$ as $\pi_1 \in \textbf{P}$. So the algorithm for solving problem $\pi_2$ runs in polynomial time. The correctness of $\pi_2$ follows from the above definition.

**Definition 18.3** Decision problem $\pi$ is NP-Complete if and only if,

1. $\pi \in \textbf{NP}$, and 
2. $\forall \pi' \in \textbf{NP}, \pi' \leq_p \pi$.

### 18.1.2 NP-complete problems

To show that problem $\pi$ is NP-complete, we need to

- show that $\pi \in \textbf{NP}$, and 
- find a $\pi^*$ that is NP-complete and show that $\pi^* \leq_p \pi$.

#### 18.1.3 3-SAT

The 3-SAT problem [CLR90] is NP-complete and this can be used to prove that other problems are NP-complete.

**Definition 18.4** The 3-SAT problem is defined as follows:

**Input:** A boolean formula $\phi$ in 3-CNF (Conjunctive Normal Form).

**Question:** Is $\phi$ satisfiable? (i.e. Is there some assignment of values to variables that makes the value of $\phi$ true?)

**Notation:** $\phi = (l_{11} \lor l_{12} \lor l_{13}) \land (l_{21} \lor l_{22} \lor l_{23}) \ldots$.

### 18.2 The Clique problem

**Definition 18.5** A clique of size $k$ in graph $G$ is a completely connected subgraph of $G$ with $k$ vertices.

Figure 3 shows an example.

![Figure 18.3: Sample graph with two cliques.](image)
Definition 18.6 The clique decision problem is:

**Input:** Graph \( G(V, E) \) and integer \( k \).

**Question:** Does \( G \) contain a clique of size \( \geq k \)?

**Theorem 18.7** Clique is \( \text{NP-complete} \).

**Proof:**

1. First prove that Clique \( \in \text{NP} \).
   
   Using the definition of \( \text{NP} \), algorithm \( A(G, y) \) verifies that \( y \) is a clique over the graph described by \( G = (V, E) \). The verification can be done in polynomial time by checking that \( y \subseteq V \), that \( \forall u, v \in y(u, v) \in E \), and that \( |y| \geq k \).

2. Next prove that 3-SAT \( \leq_p \) Clique.

   We need to find a *polynomial-time reduction* that converts an input instance \( x \) of 3-SAT to a corresponding input instance of the clique problem.

\[
\begin{align*}
\text{CNF formula } \phi & \xrightarrow{\text{polynomial time transformation}} \text{Function } f \\
& \xrightarrow{\text{Function } f} G_{\phi}, k_{\phi}
\end{align*}
\]

**Properties of the transformation:**

(a) It must be polynomial time.

(b) If \( G_{\phi} \) has a clique of size \( k_{\phi} \), then \( \phi \) must be satisfiable.

(c) If \( \phi \) is satisfiable then \( G_{\phi} \) must have a clique of size \( k_{\phi} \).

\( G_{\phi} \) has a clique of size \( k_{\phi} \) if and only if \( \phi \) is satisfiable. i.e. "YES" instances of the 3-SAT problem are mapped to "YES" instances of the clique problem and "NO" instances of 3-SAT are mapped to "NO" instances of clique.

**Proposed transformation:**

\( k_{\phi} = \) the number of clauses in \( \phi \).

\( G_{\phi} = (E, V) \) such that:

Node \( n_{ij} \in V \iff l_{ij} \in \phi \). The nodes of the graph correspond to the literals in the formula.

Edge \( \langle n_{ij}, n_{hk} \in E \) if \( i \neq h \) (the literals belong to different clauses) and \( l_{ij} \neq \neg l_{hk} \), both literals can consistently be set to true at the same time.

**Example 18.8**

\[ \phi = (a \lor \overline{b} \lor c) \land (\overline{\pi} \lor b \lor \tau) \land (\pi \lor b \lor d) \]

*Thus:* \( k_{\phi} = 3 \)

\( G_{\phi} : \)
We need to show that the properties of transformation are satisfied.

(a) The graph $G_{\phi}$ can be computed from $\phi$ in polynomial time. To compute $k_{\phi}$, we need to count the number of clauses. To create nodes of graph assign one node for each literal. To create edges, consider each possible node pair and check whether there exists an edge between the two nodes. 

(b) **Claim 18.9** If $G_{\phi}$ has a clique of size $k_{\phi}$, then $\phi$ is satisfiable.

No edges in $G_{\phi}$ connect vertices in the same clause and so each vertex in the clique corresponds to exactly one literal per clause. Also since no edges are present between a node for a literal and a node for the complement of the literal we can safely set all the literals that correspond to nodes in the clique to true. Each clause is thus true and so $\phi$ is satisfied.

(c) **Claim 18.10** If $\phi$ has a satisfying assignment, then $G_{\phi}$ has a clique of size $k_{\phi}$.

At least one literal has to be true in each clause under the satisfying assignment. The literals are consistent as they cannot be true and false at same time, i.e. $x = true$ and $\overline{x} = true$ is not possible by the nature of construction of $G_{\phi}$. Pick one true literal per clause and we form a corresponding set of nodes for those literals in $G_{\phi}$. As no node is a complement of any other node, we have edges from each node to all other nodes in the set. This gives us a clique of size $k$.

Thus, we have reduced the 3-SAT problem in polynomial time to the clique problem and the transformation used satisfies all the required properties. This concludes the proof of the theorem that clique is NP-complete.

### 18.2.1 Vertex Cover

**Definition 18.11** A vertex cover of the undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ of the vertices of $G$ such that the vertices in $V'$ touch every edge in $E$, i.e. $\forall e \in E: \exists v \in V', w \in V : e = (v, w)$.

**Example 18.12** Figure 18.5: A graph and two possible vertex covers.
**Definition 18.13** The vertex cover problem.

**Input:** A graph $G = (V, E)$, and an integer $k$.

**Question:** Does $G$ contain a vertex cover $V'$ of size $\leq k$?

**Theorem 18.14** The vertex cover problem is NP-complete.

**Proof:**

1. The vertex cover problem $\in$ NP.
   
   We can verify that a witness $W \subseteq V$ is a vertex cover of $G$ in polynomial time by inspecting each edge $e \in E$ to verify that it is touched by a vertex in $W$.

2. $3SAT \leq_p \text{VERTEX-COVER}$.

   **Idea:** Solve the satisfiability problem for any $3-CNF$ formula $\Phi$ by transforming $\Phi$ into $(G_\phi, k_\phi)$ and solving with $(G_\phi, k_\phi)$ as input the corresponding vertex cover problem.

   **The proposed transformation:**

   Set $k_\phi = V + 2m$, where $V$ is the number of variables and $m$ is the number of clauses.

   Build $G_\phi$ as follows:

   For each variable $X$ add a subgraph that looks like this:

   ![Figure 18.6: The subgraph for a variable.](image)

   For each clause $(l_1 \lor l_2 \lor l_3)$ add a subgraph that looks like this:

   ![Figure 18.7: The subgraph for a clause.](image)

   The transformation of the formula $\phi$ into $G_\phi, k_\phi$ can be performed in polynomial time since to construct $G_\phi$ only $V$ variables, $m$ clauses and the $3+2m$ literals must be considered to create and connect the corresponding nodes of $G_\phi$.

   **Example 18.15** $\Phi = (a \lor \overline{b} \lor c) \land (\overline{a} \lor \overline{b} \lor \overline{c})$

   $k_\phi = V + 2m = 3 + 2 \times 2 = 7$

   $G_\phi$: 

   ![Diagram of $G_\phi$.](image)
Claim 18.16 If $\Phi$ is satisfiable, then $G_\Phi$ has a vertex cover of size $V + 2m$.

**Proof:** If $\Phi$ is satisfiable, there must exist an assignment $A = \{x_1 = True, x_2 = False, \ldots\}$ of the variables that makes $\Phi(A)$ true. The subset $W$ of the vertices of $G_\Phi$ is now chosen as follows:

For each variable $x$ in $\Phi$, we have two vertices $x_T, x_F$ in $G_\Phi$. If $x = true$ in the assignment $A$, include $x_T$ in $W$ otherwise $x_F$.

After having done this for each variable $W$ contains $V$ vertices. These vertices cover the segments of $G_\Phi$ that correspond to variables in $\Phi$ and all the edges that connect nodes in the “variable” and “clause” components of $G_\Phi$ that correspond to true literals under the assignment $A$ in $\Phi(A)$.

For each clause $(l_1, l_2, l_3)$ in $\Phi$, we have three vertices $v_{l_1}, v_{l_2},$ and $v_{l_3}$. Since the assignment $A$ makes $\Phi(A)$ true, there must be at least one literal $l_i$ in each clause which is true. The corresponding vertex $v_{l_i}$ does not need to be included in the vertex cover $W$, because the edge from $v_{l_i}$ to the corresponding “variable” vertex $x_T$ (or $x_F$ if the $l_i = \neg x$) is already covered by $x_T$ (or $x_F$) which is in $W$. So we include the remaining vertices of the clause $v_{l_j}$ and $v_{l_k}$ in $W$. Whereby all the edges in the “clause” part of $G_\Phi$ are covered and also the yet untouched edges connecting the “variable” and “clause” components of the graph. Thus $W$ is a vertex cover which contains $V + 2m$ vertices and the above claim is proven.

Claim 18.17 If $G_\Phi$ has a vertex cover of size $V + 2m$, then $\Phi$ is satisfiable.

**Proof:** Suppose that a vertex cover $W$ of size $V + 2m$ on $G_\Phi$ is given. Looking at some variable $x$ in $\Phi$ and its corresponding vertices $x_T$ and $x_F$ in $G_\Phi$, we can see that one of these vertices $x_T$ or $x_F$ must be in $W$; otherwise the edge connecting the two vertices would not be touched by any vertex in $W$. That means $W$ has to contain at least $V$ “variable” vertices and each variable contributes at least one “variable” vertex to $W$. Looking at some clause $(l_1, l_2, l_3)$ in $\Phi$, and its corresponding vertices $v_{l_1}, v_{l_2},$ and $v_{l_3}$ in $G_\Phi$, we can see that at least two of these vertices must be in $W$, otherwise one edge of this “clause” segment of the graph would not be touched by any vertex in $W$. So $W$ has to contain at least $2m$ “clause” vertices and each clause contributes at least two “clause” vertices to $W$.

Since $W$ contains only $V + 2m$ vertices it follows that each variable contributes exactly one “variable” vertex and each clause contributes exactly two “clause” vertices to $W$. We can choose an assignment $A$ such that for each variable $x$ in $\Phi$, if $x_T \in W$, then $x = true$ in $A$ else $x = False$.

We can show via proof by contradiction that $\Phi(A)$ is true.

Assume that $\Phi(A)$ is false. If $\Phi(A)$ is false, one clause $c$ of $\Phi$ must be false under the assignment $A$. This means that all the literals in $c$ are false under the assignment $A$. In the clause $c$, we have the “clause” vertices $v_{l_1}, v_{l_2},$ and $v_{l_3}$. There are only two “clause” vertices say $v_{l_1}$ and $v_{l_2}$ in $W$. So the edge $e$ coming out of $v_{l_3}$ must be covered by the “variable” vertex at the other end of $e$. Let us now consider two cases.
**Case 1:** Let the corresponding literal \( l_3 \) to \( v_{i_3} \) be of the form \( x \), then at the other end of \( e \) the “variable” vertex is \( x_F \). Following the above argument \( x_F \) must be in the vertex cover \( W \) (otherwise \( e \) would not be touched by any vertex). But if this is true, the previously constructed assignment \( A \) has set the corresponding variable \( x = true \). Thus the literal \( l_3 = x \) is true under the assignment \( A \). This contradicts the fact that all the literals in \( c \) including \( l_3 \) were assumed to be false under the assignment \( A \).

**Case 2:** Let corresponding literal \( l_3 \) to \( v_{i_3} \) be of the form \( \overline{x} \), then at the other end of \( e \) the “variable” vertex is \( x_F \). Following the above argument \( x_F \) must be in the vertex cover \( W \) (otherwise \( e \) would not be touched by any vertex). But if this is true, the previously constructed assignment \( A \) has set the corresponding variable \( x = False \). Thus the literal \( l_3 = \overline{x} \) is true under the assignment \( A \). This contradicts the fact that all the literals in \( c \) including \( l_3 \) were assumed to be false under the assignment \( A \).

Both possible cases yield a contradiction thus \( \Phi(A) \) is true and therefore \( \Phi \) is satisfiable and the above claim is proven.

The last two proven claims conclude the proof of VERTEX-COVER being \( \text{NP} \)-complete.

### 18.2.2 The subset-sum problem

**Definition 18.18** The subset sum problem.

**Input:** A set \( S \) of integers, \( \{s_1, s_2, \ldots, s_t\} \), and a target integer \( t \).

**Question:** Does \( S \) have a subset \( S' \subseteq S \) such that the \( s_i \in S' \) sum exactly to \( t \)?

**Example 18.19** Two subset-sum problems:

- \( S = \{1, 3, 3, 7\}, t = 13 \)- Yes, such a \( S' \) exists (\( S' = \{3, 3, 7\} \)).
- \( S = \{1, 3, 3, 7\}, t = 9 \)- No subset \( S' \) exists that sums to 9.

**Theorem 18.20** The subset sum problem is \( \text{NP} \)-complete.

**Proof:**

1. The subset sum problem is in \( \text{NP} \). Verifying that a witness \( W' \subseteq S \) sums up to \( t \) can be done in polynomial time.
2. 3-\( \text{SAT} \) \( \leq_p \text{SUBSET-SUM} \).

**Proposed transformation:**

Given \( \Phi \), construct a set \( S \) of integers and a target integer \( t \) such that there is a valid sum if and only if \( \Phi \) is satisfiable.

The integers \( s_i \) (represented in the decimal system) have two parts:

\[
\begin{array}{c|c}
\# \text{variables} & \# \text{clauses} \\
\hline
\text{first part} & \text{second part} \\
\end{array}
\]

\( \Rightarrow \) number in decimal

**Figure 18.9:** The two parts of an integer \( s \).
For each variable $X_i$ two integers are constructed, $s_i$ to represent $X_i$, and $s'_i$ to represent $\bar{X}_i$. The first parts of these integers are $s_i = 10^{d-1}$ and $s'_i : 10^{d-1}$.

The second parts of these integers are constructed as follows:

- If $X_i$ is in clause $j$, the $j^{th}$ digit from the right in $S_i$ is a 1 otherwise the $j^{th}$ digit is 0.
- If $X_i$ is in clause $j$, the $j^{th}$ digit from the right in $S'_i$ is a 1 otherwise the $j^{th}$ digit is 0.

Additionally we add helper integers $h_i$ to $S$, for each clause $i$, add two copies of $10^{d-1}$.

Finally, the target integer is of the form $t = \overbrace{1111113333}^{\text{# of variables}} \overbrace{33333}^{\text{# of clauses}}$

**Example 18.21** Following is the set of integers $S$ constructed using the previous rules for the example.

$\phi = (x_1 \lor \overline{x_2} \lor x_3) \land (x_1 \lor x_2 \lor \overline{x_3}) \land (x_2 \lor \overline{x_3} \lor x_4)t = 11113333$ (because we have 4 variables and 3 clauses)

<table>
<thead>
<tr>
<th>Set 'S'</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>1001</td>
</tr>
<tr>
<td>$S'_1$</td>
<td>1010</td>
</tr>
<tr>
<td>$S_2$</td>
<td>10110</td>
</tr>
<tr>
<td>$S'_2$</td>
<td>10001</td>
</tr>
<tr>
<td>$S_3$</td>
<td>100001</td>
</tr>
<tr>
<td>$S'_3$</td>
<td>100100</td>
</tr>
<tr>
<td>$S_4$</td>
<td>1000100</td>
</tr>
<tr>
<td>$S'_4$</td>
<td>1000010</td>
</tr>
<tr>
<td>$h_1$</td>
<td>1</td>
</tr>
<tr>
<td>$h_2$</td>
<td>1</td>
</tr>
<tr>
<td>$h_3$</td>
<td>10</td>
</tr>
<tr>
<td>$h_4$</td>
<td>10</td>
</tr>
<tr>
<td>$h_5$</td>
<td>100</td>
</tr>
<tr>
<td>$h_6$</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 18.1: Reduction from CNF to a Subset sum problem. $S_i$'s are generated using the rules for construction of the first and second parts of each integer. $h_i$'s are the helper integers required in the set $S$.

Now we need to show that this construction given $\Phi$ yields a set $S$ of integers and a target integer $t$ such that:

**Claim 18.22** The following are equivalent:

1. There is a subset $S'$ that solves the subset problem $(S,t)$.
2. $\Phi$ is satisfiable.

**Proof:** Let $v$ be the number of variables and $m$ the number of clauses in $\Phi$. 
• \( \Rightarrow \): Say for \((S, t)\) a subset \(S'\) is given such that its elements sum exactly up to \(t\). In order to equate the first part of \(t\), \(S'\) must contain for all \(i = 1, 2, \ldots, v\) either \(s_i\) or \(s'\). Otherwise the \(i^{th}\) digit of the first part of \(t\) would be 2, if both \(s_i\) and \(s'\) were in \(S'\), or 0, if none of them were in \(S'\). So by inspecting if \(s_i\) or \(s'\) is in \(S'\), we can construct an assignment \(A\) for the formula \(\Phi\): if \(s_i\) is in \(S'\) then \(X_i = true\), otherwise \((s' \in S') X_i = false\).

We now show via proof by contradiction that \(\Phi(A)\) must be \textit{true}.

Assume that \(\Phi(A)\) is \textit{false}. Then one clause (say it is the \(c^{th}\) clause) in \(\Phi\) must be \textit{false}. This corresponds to the situation that in \(S'\) no integer \(s_i\) or \(s'\) is included that has a 1 at its \(c^{th}\) digit in its second part. (This is true because if such an \(s_i\) existed the assignment \(A\) would dictate that \(X_i = true\) and the \(c^{th}\) digit being 1 means after the construction that the literal \(X_i\) is in the \(c^{th}\) clause. But with \(X_i = true\) under \(A\), we have that the literal \(X_i\) in the \(c^{th}\) clause is \textit{true}. Thus the \(c^{th}\) clause is \textit{true}\) under \(A\). This contradicts the fact that it was just assumed to be \textit{false}\) under \(A\). A similar argument holds if there exists an \(s'\) such that it has a 1 at its \(c^{th}\) digit in its second part.) But if there is no integer \(s_i\) or \(s'\) having a 1 at the \(c^{th}\) digit in its second part, the only integers of \(S'\) that can contribute to the \(c^{th}\) digit of \(t\) in its second part are the two helper variables \(h_c\) and \(h'^c\). But these two can only contribute in total 2 to the \(c^{th}\) digit of \(t\)'s second part, and the \(c^{th}\) digit of \(t\) must be equal to 3. So the subset \(S'\) cannot add up to \(t\) since the construction does not allow any carries. This contradicts the fact that \(S'\) was given to be a valid solution to the subset sum problem.

Thus \(\Phi(A)\) must be \textit{true}.

• \( \Leftarrow \): \(\Phi\) is satisfiable. This means that there is a truth assignment \(A\) of the variables in \(\Phi\) such that \(\Phi(A)\) is \textit{true}. We include in the subset \(S'\) for each variable \(X_i\) in \(\Phi\) either \(s_i\), if \(X_i = true\) under \(A\), or \(s'\), if \(X_i = false\). This guarantees that the first part of the integers in \(S'\) sum up correctly as no integers have the same digits set to 1 except for \(s_i\) and \(s'\) in the first part. Since \(\Phi(A)\) is \textit{true}, each clause of \(\Phi\) must be \textit{true} under \(A\). This means that for every clause \(c = 1, 2, \ldots, m\), there must be at least one integer \(s_i\) or \(s'\) in the so far constructed \(S'\) subset such that the \(c^{th}\) digit in its second part is 1. Thus for each clause \(c\), we can add zero, one or both helper variables \(h_c, h'^c\) to \(S'\) to equal the required 3 at the \(c^{th}\) digit in the second part of \(t\). Thus adding all selected integers \((s_i \text{ or } s'_i)\) and if necessary the helper integers for each clause to \(S'\), we have constructed a valid solution subset \(S'\) for the subset problem \((S, t)\).

\(\blacksquare\)

This concludes the proof.

\(\blacksquare\)

### 18.3 Reference