Reduction

If a Boolean expression $\phi$ in 3CNF has $n$ variables, and $m$ clauses, we construct for each variable $v$ the following gadget.

![Diagram of a gadget for a variable]

In addition, for every clause $c$, we have two elements $x_c$ and $y_c$. If the literal $v$ occurs in $c$, we include the triple $(x_c, y_c, z_{vc})$ in $M$.

Similarly, if $\neg v$ occurs in $c$, we include the triple $(x_c, y_c, z_{vc})$ in $M$.

Finally, we include extra dummy elements in $X$ and $Y$ to make the numbers match up.

Exact Set Covering

Two other well-known problems are proved NP-complete by immediate reduction from 3DM.

**Exact Cover by 3-Sets** is defined by:

Given a set $U$ with $3n$ elements, and a collection $S = \{S_1, \ldots, S_m\}$ of three-element subsets of $U$, is there a sub-collection containing exactly $n$ of these sets whose union is all of $U$?

The reduction from 3DM simply takes $U = X \cup Y \cup Z$, and $S$ to be the collection of three-element subsets resulting from $M$.

Set Covering

More generally, we have the **Set Covering** problem:

Given a set $U$, a collection of $S = \{S_1, \ldots, S_m\}$ subsets of $U$ and an integer budget $B$, is there a collection of $B$ sets in $S$ whose union is $U$?
Knapsack

**Knapsack** is a problem which generalises many natural scheduling and optimisation problems, and through reductions has been used to show many such problems **NP**-complete.

In the problem, we are given $n$ items, each with a positive integer value $v_i$ and weight $w_i$.

We are also given a maximum total weight $W$, and a minimum total value $V$.

Can we select a subset of the items whose total weight does not exceed $W$, and whose total value exceeds $V$?

Reduction

The proof that **Knapsack** is **NP**-complete is by a reduction from the problem of Exact Cover by 3-Sets.

Given a set $U = \{1, \ldots, 3n\}$ and a collection of 3-element subsets of $U$, $S = \{S_1, \ldots, S_m\}$.

We map this to an instance of **Knapsack** with $m$ elements each corresponding to one of the $S_i$, and having weight and value

$$\sum_{j \in S_i} (m + 1)^j$$

and set the target weight and value both to

$$\sum_{j=0}^{3n-1} (m + 1)^j$$

Scheduling

Some examples of the kinds of scheduling tasks that have been proved **NP**-complete include:

**Timetable Design**

Given a set $H$ of *work periods*, a set $W$ of *workers* each with an associated subset of $H$ (available periods), a set $T$ of *tasks* and an assignment $r : W \times T \to \mathbb{N}$ of *required work*, is there a mapping $f : W \times T \times H \to \{0, 1\}$ which completes all tasks?

**Sequencing with Deadlines**

Given a set $T$ of *tasks* and for each task a *length* $l \in \mathbb{N}$, a release time $r \in \mathbb{N}$ and a deadline $d \in \mathbb{N}$, is there a work schedule which completes each task between its release time and its deadline?

**Job Scheduling**

Given a set $T$ of *tasks*, a number $m \in \mathbb{N}$ of processors a length $l \in \mathbb{N}$ for each task, and an overall deadline $D \in \mathbb{N}$, is there a multi-processor schedule which completes all tasks by the deadline?
Responses to NP-Completeness

Confronted by an NP-complete problem, say constructing a timetable, what can one do?

- It’s a single instance, does asymptotic complexity matter?
- What’s the critical size? Is scalability important?
- Are there guaranteed restrictions on the input? Will a special purpose algorithm suffice?
- Will an approximate solution suffice? Are performance guarantees required?
- Are there useful heuristics that can constrain a search? Ways of ordering choices to control backtracking?