NP-COMPLETE PROBLEMS

- Characterizing **NP**
- Variants of satisfiability
- Graph-theoretic problems
- Coloring problems
- Sets and numbers
- Pseudopolynomial algorithms

(C. Papadimitriou: *Computational complexity*, Chapter 9)

1. Characterizing NP

Definition.

1. A relation $R \subseteq \Sigma^* \times \Sigma^*$ is *polynomially decidable* iff there is a deterministic TM deciding the language $\{x;y \mid (x,y) \in R\}$ in polynomial time.
2. A relation R is *polynomially balanced* if $(x,y) \in R$ implies $|y| \leq |x|^k$ for some $k \geq 1$.

Proposition. Let $L \subseteq \Sigma^*$ be a language.

Now $L \in \mathbf{NP}$ iff there is a polynomially balanced and polynomially decidable relation R such that $L = \{x \in \Sigma^* \mid (x,y) \in R \text{ for some } y \in \Sigma^*\}$.

Proof

($\Leftarrow$) Suppose there is such a relation R. Then L is decided by a NTM that on input x, guesses a y of length at most $|x|^k$ and uses the machine for R to decide in polynomial time whether $(x,y) \in R$.

($\Rightarrow$) Suppose that $L \in \mathbf{NP}$, i.e. there is a NTM N deciding L in time $|x|^k$ for some k.

Define a relation R as follows: $(x,y) \in R$ iff y is the encoding of an accepting computation of N on input x. Now R is polynomially

- balanced (each computation of N is polynomially bounded) and
- decidable (since it can be checked in linear time whether y encodes an accepting computation of N on x).
- As N decides L, $L = \{x \mid (x,y) \in R \text{ for some } y\}$. $\square$

Succinct certificates

A problem is in **NP** if any "yes" instance x of the problem has at least one **succinct certificate**, or polynomial witness, y. **NP** contains a huge number of practically important, natural computational problems:

- A typical problem is to construct a mathematical object satisfying certain specifications (path, solution of equations, routing, VLSI layout,...). This is the certificate.
- The decision version of the problem is determine whether at least one such an object exists for the input.
- The object is usually not very large compared to the input.
- The specifications of the object are usually simple enough to be checkable in polynomial time.

Boundary between NP and P

- Most problems arising in computational practice are in **NP**.
- Computational complexity theory provides us tools to study which problems in **NP** belong to **P** and which do not.
- **NP**-completeness is a basic tool in this respect:
  Showing that a problem is **NP**-complete implies that the problem is among the least likely to be in **P**.
  (If an **NP**-complete problem is in **P**, then $\mathbf{NP} = \mathbf{P}$.)

NP-completeness and algorithm design techniques

When a problem is known to be **NP**-complete, further efforts are usually directed to:

(i) Attacking special cases
(ii) Approximation algorithms
(iii) Studying average case performance
(iv) Randomized algorithms
(v) (Exponential) algorithms that are practical for small instances
(vi) Local search methods

2. Variants of Satisfiability

- Many problems if generalized enough become **NP**-complete.
- Often it is important to find the dividing line between **P** and **NP**-completeness.
- One basic technique is to investigate the set of instances produced by a reduction R involved in the **NP**-completeness proof in order to capture another **NP**-complete problem.
- Next we consider variants of SAT such as

  3SAT, 2SAT, MAX2SAT, and NAESAT

  and analyze their computational complexities.

kSAT problems

Definition. kSAT, where $k \geq 1$ is an integer, is the set of Boolean expressions $\phi \in$ SAT (in CNF) whose all clauses have exactly k literals.

Proposition. 3SAT is **NP**-complete.

Proof.

- 3SAT is in **NP** as a special case of SAT which is in **NP**.
- CIRCUIT SAT was shown to be **NP**-complete and a reduction from CIRCUIT SAT to SAT has already been presented.
- Consider now the clauses in the reduction. They have all at most 3 literals. Each clause with one or two literals can be modified to an equivalent clause with exactly 3 literals by duplicating literals.
- Hence, we can reduce CIRCUIT SAT to 3SAT. □

Narrowing NP-complete languages

- An **NP**-complete languages can sometimes be narrowed down by *transformations* which eliminate certain features of the language but still preserve **NP**-completeness.
- The following result is a typical example.

Proposition. 3SAT remains **NP**-complete even if each variable is restricted to appear at most three times in a Boolean expression $\phi \in$ 3SAT and each literal at most twice in ϕ.

Proof. This is shown by a reduction where any instance ϕ of 3SAT is rewritten to eliminate the forbidden features.

Proof

- Consider a variable x appearing $k > 3$ times in ϕ.
  (i) Replace the first occurrence of x in ϕ by x_1, the second by x_2, and so on where $x_1, \ldots, x_k$ are new variables.
  (ii) Add clauses $(\neg x_1 \vee x_2), (\neg x_2 \vee x_3), \ldots, (\neg x_k \vee x_1)$ to ϕ.
- Let ϕ' be the expression ϕ modified systematically in this way.
- It follows that ϕ' has the desired syntactic properties.
- Now ϕ is satisfiable iff ϕ' is satisfiable:
  For each x appearing $k > 3$ times in ϕ, the truth values of $x_1, \ldots, x_k$ are the same in each truth assignment satisfying ϕ'. □

Boundary between P and NP-completeness

- The boundary is between 2SAT and 3SAT.
- For an instance ϕ of 2SAT, there is a polynomial time algorithm which is based on reachability in a graph associated with ϕ.

Definition. Let ϕ be an instance of 2SAT.

Define a graph $G(\phi)$ as follows:

– The variables of ϕ and their negations form the vertices of $G(\phi)$.

– There is an arc (α, β) iff there is a clause $\overline{\alpha} \vee \beta$ or $\beta \vee \overline{\alpha}$ in ϕ, i.e., if (α, β) is an arc, so is $(\overline{\beta}, \overline{\alpha})$ where $\overline{\alpha}$ is the complement of α.

Theorem. Let ϕ be an instance of 2SAT.

Then ϕ is unsatisfiable iff there is a variable x such that there are paths from x to $\neg x$ and from $\neg x$ to x in $G(\phi)$.

The complexity of 2SAT—cont'd

Corollary. 2SAT is in **NL** ($\subseteq$ **P**).

Proof. Since **NL** is closed under complement, it is sufficient to show that 2SAT COMPLEMENT is in **NL**.

The reachability condition of the preceding theorem can be tested in logarithmic space non-deterministically by guessing a variable x and paths from x to $\neg x$ and back. □

MAX2SAT is a generalization of 2SAT:

INSTANCE: a Boolean expression ϕ in CNF (having at most two literals per clause) and an integer bound K.
QUESTION: Is there a truth assignment satisfying at least K clauses?

Theorem. MAX2SAT is **NP**-complete. (See tutorials.)

The case of not-all-equal SAT (NAESAT)

For each $\phi \in$ NAESAT $\subset$ 3SAT, there is a truth assignment so that the three literals in each clause of ϕ do not have the same truth value.

Theorem. NAESAT is **NP**-complete.

Proof.

- CIRCUIT SAT was shown to be **NP**-complete and a reduction R from CIRCUIT SAT to SAT has already been presented.
- For all one- and two-literal clauses in the reduced circuit $R(C)$, add the same literal, say z, to make them 3-literal clauses.

Claim: it holds for the resulting Boolean expression $R_z(C)$ in 3CNF:

$$R_z(C) \in \text{NAESAT} \ \textit{iff} \ R(C) \in \text{SAT iff } C \in \text{CIRCUIT SAT}.$$

(The latter iff-relationship is already known.)

($\Rightarrow$) If a truth assignment T satisfies $R_z(C)$ in the sense of NAESAT, so does the complementary truth assignment $\overline{T}$.

Thus z is **false** in either T or $\overline{T}$ which implies that $R(C)$ is satisfied by T or $\overline{T}$. Thus C is satisfiable.

($\Leftarrow$) If C is satisfiable, then there is a truth assignment T satisfying $R(C)$. Let us then extend T for $R_z(C)$ by assigning $T(z) = \textbf{false}$.

In no clause of $R_z(C)$ all literals are **true** (they cannot be all **false**):

(i) Clauses for **true**/**false**/NOT/variable gates contain z that is **false**.

(ii) For AND gates the clauses are: $(\neg g \vee h \vee z)$, $(\neg g \vee h' \vee z)$, $(g \vee \neg h \vee \neg h')$ where in the first two z is **false**, and in the third all three cannot be true as then the first two would be not true.

(iii) The case of OR gates is very similar. □

3. Graph-Theoretic Problems

- In this section, we will consider only undirected graphs $G = (V, E)$ and their properties which lead to computational problems.
- For instance, consider the problem of finding an *independent* subset I of V: for all $i, j \in I$ there is no edge $[i, j] \in E$.

INDEPENDENT SET:
INSTANCE: An undirected graph $G = (V, E)$ and an integer K.
QUESTION: Is there an independent set $I \subseteq V$ with $|I| = K$.

Theorem. INDEPENDENT SET is **NP**-complete. (See tutorials.)

The subclass of graphs needed in the reduction implies the following:

Corollary. 4-DEGREE INDEPENDENT SET is **NP**-complete.

Graph problems: CLIQUE and NODE COVER

- The problems in graph theory are often closely related; suggesting even trivial reductions between problems.

Example. Consider the following two graph theoretic problems:

CLIQUE:
INSTANCE: An undirected graph $G = (V, E)$ and an integer K.
QUESTION: Is there a clique $C \subseteq V$ with $|C| = K$?
(A set $C \subseteq V$ is clique iff for all $i, j \in I$, $[i, j] \in E$)

NODE COVER:
INSTANCE: An undirected graph $G = (V, E)$ and an integer B.
QUESTION: Is there a set $C \subseteq V$ with $|C| \leq B$ such that for all $[i, j] \in E$, $i \in C$ or $j \in C$?

Trivial reductions for CLIQUE and NODE COVER

- Independent sets are closely related to cliques and node covers:

  A set $I \subseteq V$ of vertices is

  1. an independent set of G iff it is a clique of the *complement* graph $\overline{G}$, and
  2. an independent set of G iff $V - I$ is a node cover of G.

- Thus an instance $G; K$ of INDEPENDENT SET can be reduced to

  – an instance $\overline{G}; K$ of CLIQUE, and
  – an instance $G; |V| - K$ of NODE COVER.

Corollary. CLIQUE and NODE COVER are **NP**-complete.

Graph problems: MIN CUT and MAX CUT

- A cut in an undirected graph $G = (V, E)$ is a partition of the nodes into two nonempty sets S and $V - S$.
- The size of a cut is the number of edges between S and $V - S$.
- The problem of finding a cut with the smallest size is in **P**:

  (i) The smallest cut that separates two given nodes s and t equals to the maximum flow from s to t.

  (ii) Minimum cut: find the maximum flow between a fixed s and all other nodes and choose the smallest value found.

- However, the problem of deciding whether there is a cut of a size greater than or equal to K (MAX CUT) is much harder:

Theorem. MAX CUT is **NP**-complete.

Reduction from NAESAT to MAX CUT

The **NP**-completeness of MAX CUT is shown for graphs with multiple edges between nodes by a reduction from NAESAT.

- For a conjunction of clauses $\phi = C_1 \wedge \ldots \wedge C_m$, we construct a graph $G = (V, E)$ so that

  G has a cut of size 5m iff ϕ is satisfied in the sense of NAESAT.

- The nodes of G are $x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n$ where $x_1, \ldots, x_n$ are the variables in ϕ.
- The edges in G include a triangle $[\alpha, \beta, \gamma]$ for each clause $\alpha \vee \beta \vee \gamma$ and n_i copies of the edge $[x_i, \neg x_i]$ where n_i is the number of occurrences of x_i or $\neg x_i$ in the clauses.

Correctness of the reduction

- Suppose there is a cut $(S, V - S)$ of size $5m$ or more.
- All variables can be assumed separate from their negations:

  If both $x_i, \neg x_i$ are on the same side, they contribute at most $2n_i$ edges to the cut and, hence, changing the one with fewer neighbors does not decrease the size of the cut.

- Let S be the set of true literals and $V - S$ those false.
- The total number of edges in the cut joining opposite literals is $3m$. The remaining $2m$ are coming from triangles meaning that all m triangles are cut, i.e. ϕ is satisfied in the sense of NAESAT.
- Conversely, a satisfying truth assignment (in the sense of NAESAT) gives rise to a cut of size $5m$. $\square$

Graph problems: MAX BISECTION

- In applications of graph partitioning, the sizes of S and $V-S$ cannot be arbitrarily small or large.

  MAX BISECTION is the problem of determining whether there is a cut $(S,V-S)$ with size of K or more such that $|S| = |V-S|$.
- Is MAX BISECTION easier than MAX CUT?

Lemma. MAX BISECTION is **NP**-complete.

Proof. Reducing MAX CUT to MAX BISECTION by modifying input:

Add $|V|$ disconnected new nodes to G. Now every cut of G can be made a bisection by appropriately splitting the new nodes.

Now $G=(V,E)$ has a cut $(S,V-S)$ with size of K or more iff the modified graph has a cut with size of K or more with $|S| = |V-S|$. $\square$

Graph problems: BISECTION WIDTH

- However, the respective minimization problem, i.e. MIN CUT with the bisection requirement, is **NP**-complete.
- BISECTION WIDTH: is there a bisection of size K or less?

Theorem. BISECTION WIDTH is **NP**-complete.

Proof. A reduction from MAX BISECTION:

A graph $G=(V,E)$ where $|V| = 2n$ for some n has a bisection of size K or more iff the complement $\overline{G}$ has a bisection of size n^2-K or less. $\square$

General instructions how to establish NP-completeness

Designing an **NP**-completeness proof for a given problem Q

- Work on small instances of Q to develop gadgets/primitives.
- Look at known **NP**-complete problems.
- Design a reduction R **from** *a known **NP**-complete problem* **to** Q.
- Typical ingredients of a reduction:

  choices + consistency + constraints.
- The key question is how to express these in Q?

Graph problems: HAMILTON PATH

Theorem. HAMILTON PATH is **NP**-complete.

Proof.

- Reduction from 3SAT to HAMILTON PATH:

  given a formula ϕ in CNF with variables $x_1,\ldots,x_n$ and clauses $C_1,\ldots,C_m$ each with three literals, we construct a graph $R(\phi)$ that has a Hamilton path iff ϕ is satisfiable.
- *Choice gadgets* select a truth assignment for variables x_i.
- *Consistency gadgets* (XOR) enforce that all occurrences of x_i have the same truth value and all occurrences of $\neg x_i$ the opposite.
- *Constraint gadgets* guarantee that all clauses are satisfied.

Reduction from 3SAT to HAMILTON PATH

The graph $R(\phi)$ is constructed as follows:

- The *choice gadgets* of variables x_i are connected in series.
- A *constraint gadget* (triangle) for each clause with an edge identified with each literal l in the clause.
  - If l is x_i, then XOR to **true** edge of choice gadget of x_i.
  - If it is $\neg x_i$, then XOR to **false** edge of choice gadget of x_i.
- All nodes of the triangles, the end node of choice gadgets and a new node 3 form a clique. Add a node 2 connected to 3.

Basic idea: each side of the constraint gadget is traversed by the Hamilton path iff the corresponding literal is **false**. Hence, at least one literal in any clause is **true** since otherwise all sides for its triangle should be traversed which is impossible (implying no Hamilton path).

Correctness of the reduction

- If there is a Hamilton path, ϕ is satisfiable:

  The path starts at 1, makes a truth assignment, traverses the triangles in some order and ends up in 2. The truth assignment satisfies ϕ as there are no triangle where all sides are traversed, i.e., all literals **false**.

- If ϕ is satisfiable, there is a Hamilton path:

  From a satisfying truth assignment, we construct a Hamilton path by starting at 1, traversing choice gadgets according to the truth assignment, the rest is a big clique for which a trivial path can be found leading to 3 and then to 2. □

Travelling salesperson (TSP) revisited

Corollary. TSP(D) is **NP**-complete.

Proof: A reduction from HAMILTON PATH to TSP(D). Given a graph G with n nodes, construct a distance matrix d_{ij} and a budget B so that there is a tour of length B or less iff G has a Hamilton path.

- There are n cities and the distance $d_{ij} = 1$ if there is $[i, j] \in G$ and $d_{ij} = 2$ otherwise. The budget $B = n+1$.
- If there is a tour of length $n+1$ or less, then there is at most one pair $(\pi(i), \pi(i+1))$ in it with cost 2, i.e., a pair for which $[\pi(i), \pi(i+1)]$ is not an edge. Removing it gives a Hamilton path.
- If G has a Hamilton path, then its cost is $n-1$ and it can be made a tour with additional cost of 2. □

4. Coloring Problems

Consider the following problem:

k-COLORING:
INSTANCE: An undirected graph $G = (V, E)$.
QUESTION: Is there an assignment of one of k colors to each of the nodes in V such that any two nodes i, j connected by an edge $[i, j] \in E$ do not have the same color?

☞ Coloring with $k = 2$ colors is easy (in **P**) but not when $k = 3$.

Determining the complexity of 3-COLORING

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

Proof. A reduction from NAESAT to 3-COLORING.

- For a conjunction clauses $\phi = C_1 \wedge \ldots \wedge C_m$ with variables $x_1, \ldots, x_n$, construct a graph $G(\phi)$ that can be colored with $\{0,1,2\}$ iff there is a truth assignment satisfying all clauses in the way of NAESAT.
- *Choice gadgets:* For each variable x_i, we introduce a triangle $[a, x_i, \neg x_i]$, i.e. all triangles share a node a.
- *Constraints:* For each clause C_i: a triangle $[C_{i1}, C_{i2}, C_{i3}]$ where each C_{ij} is further connected to the node with the jth literal in C_i.

Correctness of the reduction

($\Rightarrow$) Suppose G can be colored with $\{0,1,2\}$ and a has color 2. This induces a truth assignment T via the colors of the nodes x_i: if the color is 1, then $T(x_i) = \mathbf{true}$ else $T(x_i) = \mathbf{false}$.

If we assume that T assigns all literals of some clause C_i to true/false, then color 1/0 cannot be used for coloring $[C_{i1}, C_{i2}, C_{i3}]$, a contradiction. Thus ϕ is satisfied in the sense of NAESAT.

($\Leftarrow$) Assume that ϕ is satisfied by T in the sense of NAESAT. Then we can extract a coloring for G from T as follows:

1. Node a is colored with color 2.
2. If $T(x_i) = \mathbf{true}$, then color x_i with 1 and $\neg x_i$ with 0 else vice versa.
3. From each $[C_{i1}, C_{i2}, C_{i3}]$, color two literals having opposite truth values with 0 (**true**) and 1 (**false**). Color the third with 2. □

5. Sets and Numbers

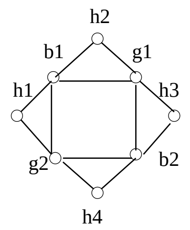
TRIPARTITE MATCHING:
INSTANCE: Three sets B (boys), G (girls), and H (homes) each containing n elements and a ternary relation $T \subseteq B \times G \times H$.
QUESTION: Is there a set of n triples in T no two of which have a component in common?

Theorem. TRIPARTITE MATCHING is **NP**-complete.

Proof. By a reduction from 3SAT. Each variable x has a combined choice and consistency gadget and each clause c a dedicated pair of a boy b_c and a girl g_c and three triples (b_c, g_c, h_l) where h_l ranges over the three homes corresponding to the occurrences of literals in the clause (appearing in the combined gadgets).

The combined gadget for choice and consistency

The gadget for a variable x involves k boys and k girls forming a k circle and $2k$ homes where k is either the number of occurrences of x or its negation whichever is larger. (Recall that k can be assumed to be 1 or 2). The case $k = 2$ is given alongside.



- Each occurrence of x is represented by homes h_{2i-1} and those of $\neg x$ by homes h_{2i}.
- Exactly two kinds of matchings possible:
  - "$T(x) = \mathbf{true}$": b_i with g_i and h_{2i}.
  - "$T(x) = \mathbf{false}$": b_i with g_{i-1} (g_k if $i = 1$) and h_{2i-1}.

Correctness of the reduction

- Note that "$T(x) = \mathbf{true}$" matching leaves the homes for x unoccupied and "$T(x) = \mathbf{false}$" those for $\neg x$ unoccupied.
- For a clause c, the dedicated b_c and g_c are matched to a home that is left unoccupied when the variables are assigned truth values meaning that it corresponds to a **true** literal satisfying c.
- One more detail needs to be settled: there are more homes H than boys B and girls G (but $|B| = |G|$).
- Add $l = |H| - |B|$ new boys and l new girls. The ith such girl participates in $|H|$ triples with the ith boy and every home.
- Now a tripartite matching exists iff the set of clauses is satisfiable.

Other problems involving sets

1. SET COVERING:
   INSTANCE: A family $F = \{S_1, \ldots, S_n\}$ of subsets of a finite set U and an integer B.
   QUESTION: Is there a set of B sets in F whose union is U?
2. SET PACKING:
   INSTANCE: A family $F = \{S_1, \ldots, S_n\}$ of subsets of a finite set U and an integer K.
   QUESTION: Is there a set of K pairwise disjoint sets in F?
3. EXACT COVER BY 3-SETS:
   INSTANCE: A family $F = \{S_1, \ldots, S_n\}$ of subsets of a finite set U such that $|U| = 3m$ for some integer m and for all i $|S_i| = 3$.
   QUESTION: Is there a set of m sets in F that are disjoint and have U as their union?

Classifications obtained by generalization

Corollary. SET COVERING, SET PACKING, and EXACT COVER BY 3-SETS are all **NP**-complete.

- TRIPARTITE MATCHING can be reduced to EXACT COVER BY 3-SETS by noticing that it is a special case where U is partitioned in three sets B, G, H with $|B| = |G| = |H|$ and $F = \{\{b, g, h\} \mid (b, g, h) \in T\}$.
- EXACT COVER BY 3-SETS can be reduced to SET COVERING as a special case where the universe has $3m$ elements, all sets in F have 3 elements and the budget $B = m$.
- EXACT COVER BY 3-SETS can be reduced to SET PACKING as a special case where the universe has $3m$ elements, all sets in F have 3 elements and limit $K = m$. $\square$

A number problem: INTEGER PROGRAMMING

INSTANCE: a system of linear inequalities with integer coefficients.
QUESTION: Is there an integer solution of the system?

Corollary. INTEGER PROGRAMMING is **NP**-complete.

Proof. SET COVERING reducible to INTEGER PROGRAMMING:

Given a family $F = \{S_1, \ldots, S_n\}$ of subsets of a finite set $U = \{u_1, \ldots, u_m\}$ and a integer B, construct a system:

$$\begin{array}{ll} 0 \le x_1 \le 1, \ldots, 0 \le x_n \le 1 & a_{11}x_1 + \cdots + a_{1n}x_n \ge 1 \\ \Sigma_{i=1}^{n} x_i \le B & \vdots \\ & a_{m1}x_1 + \cdots + a_{mn}x_n \ge 1 \end{array}$$

where $a_{ij} = 1$ if ith element of U is in the set S_j, otherwise $a_{ij} = 0$. (The idea: $x_i = 1$ if S_i in the cover and otherwise $x_i = 0$.) $\square$

Further problems involving numbers

1. LINEAR PROGRAMMING (i.e. INTEGER PROGRAMMING where non-integer solutions are allowed) is in **P**.
2. KNAPSACK:
   INSTANCE: A set of n items with each item i having a value v_i and a weight w_i (both positive integers) and integers W and K.
   QUESTION: Is there a subset S of the items such that $\Sigma_{i \in S} w_i \leq W$ but $\Sigma_{i \in S} v_i \geq K$?

Theorem. KNAPSACK is **NP**-complete.

Proof. We show that a simple special case of KNAPSACK is **NP**-complete where $v_i = w_i$ for all i and $W = K$:

INSTANCE: A set of integers $w_1, \ldots, w_n$ and an integer K.
QUESTION: Is there a subset S of the integers with $\Sigma_{i \in S} w_i = K$?

Reduction from EXACT COVER BY 3-SETS

The reduction is based on the set $U = \{1, 2, \ldots, 3m\}$ and the sets $S_1, \ldots, S_n$ given as bit vectors $\{0,1\}^{3m}$ and $K = 2^{3m} - 1$. Then the task is to find a subset of bit vectors that sum to K.

$$\begin{array}{c|ccccc} \rightarrow & 0 & 1 & \ldots & 0 & 0 \\ & 1 & 0 & \ldots & 0 & 0 \\ & \vdots & & & & \\ \rightarrow & 0 & 0 & \ldots & 1 & 1 \\ \hline & 1 & 1 & \ldots & 1 & 1 \end{array}$$

- This does not quite work because of the carry bit, but the problem can be circumvented by using $n+1$ as the base rather than 2.
- Now S_i corresponds to $w_i = \Sigma_{j \in S_i} (n+1)^{3m-j}$.
- Then a set of these integers w_i adds up to $K = \Sigma_{j=0}^{3m-1} (n+1)^j$ iff there is an exact cover among $\{S_1, S_2, \ldots, S_n\}$. □

6. Pseudopolynomial Algorithms

Proposition. Any instance of KNAPSACK can be solved in $O(nW)$ time where n is the number of items and W is the weight limit.

Proof.

- Define $V(w,i)$: the largest value attainable be selecting some among the first i items so that their total weight is exactly w.
- Each $V(w,i)$ with $w = 1, \ldots, W$ and $i = 1, \ldots, n$ can be computed by
  $$V(w, i+1) = \max\{V(w,i), v_{i+1} + V(w - w_{i+1}, i)\}$$
  where $V(w,0) = 0$ for all w and $V(w,i) = -\infty$ if $w \leq 0$.
- For each entry this can be done in constant number of steps and there are nW entries. Hence, the algorithm runs in $O(nW)$ time.
- An instance is answered "yes" iff there is an entry $V(w,i) \geq K$. □

Strong NP-completeness

- The preceding algorithm is not polynomial w.r.t. the length of the input (which is $O(n \log W)$) but exponential ($W = 2^{\log W}$).
- An algorithm where the time bound is polynomial in the integers in the input (not their logarithms) is called pseudopolynomial.
- A problem is called **strongly NP-complete** if the problem remains **NP**-complete even if any instance of length n is restricted to contain integers of size at most $p(n)$, for a polynomial p.
  ☞ Strongly **NP**-complete problems cannot have pseudopolynomial algorithms (unless **P** = **NP**).
- SAT, MAX CUT, TSP(D), HAMILTON PATH, … are strongly **NP**-complete but KNAPSACK is not.

Yet another number problem: BIN PACKING

INSTANCE: N positive integers $a_1, \ldots, a_N$ (items) and integers C (capacity) and B (number of bins).
QUESTION: Is there a partition of the numbers into B subsets such that for each subset S, $\Sigma_{a_i \in S} a_i \leq C$?

- BIN PACKING is strongly **NP**-complete:

  Even if the integers are restricted to have polynomial values (w.r.t. the length of input), BIN PACKING remains **NP**-complete.

- Any pseudopolynomial algorithm for BIN PACKING would yield a polynomial algorithm for all problems in **NP** implying $\mathbf{P} = \mathbf{NP}$.

Learning Objectives

- The concept of **NP**-completeness and its characterizations in terms of succinct certificates.
- You should know basic techniques to prove problems **NP**-complete and be able to construct such proofs on your own.
- A basic repertoire of **NP**-complete problems (related with satisfiability, graphs, sets, and numbers) to be used in further **NP**-completeness proofs.
- The definition of strong **NP**-completeness and awareness of number problems which are (not) strongly **NP**-complete.