NP-COMPLETE PROBLEMS

An alternative characterization of **NP**:

➤ 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.

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

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 $L \in \mathbf{NP}$. Hence, there is a NTM N deciding L in time $|x|^k$ for some k.

Define R: $(x,y) \in R$ iff y is the encoding of an accepting computation of N on input x.

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

Problems in NP have 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: tools to study which problems in **NP** belong to **P** and which do not.

➤ **NP**-completeness is a basic tool:

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 complements algorithm design techniques

When a problem is shown **NP**-complete, 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

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.
- A basic technique is to investigate the subclass of the problem that is needed in the **NP**-completeness proof to capture another **NP**-complete problem.
- Next we consider the SAT problem as an example.

SAT problems

kSAT is the subclass of SAT where all clauses have 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 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 a literal once or twice.
- Hence, we can reduce CIRCUIT SAT to 3SAT implying that 3SAT is **NP**-complete. □

SAT—cont'd

An **NP**-complete subclass can sometimes be narrowed down by *transformations* which eliminate certain features but still preserve **NP**-completeness.

The following result is a typical example.

Proposition. The subclass of SAT where each clause has at most 3 literals is **NP**-complete even if each variable is restricted to appear at most three times, and each literal at most twice.

Proof.

- This is shown by a reduction where any instance of 3SAT is rewritten to eliminate the forbidden features.
- Consider a variable x appearing $k > 3$ times in the expression ϕ.
  (i) Replace the first appearance of x by x_1, the second by x_2, and so on where $x_1, \ldots, x_k$ are new variables.
  (ii) Add clauses $(\neg x_1 \vee x_2), (\neg x_2 \vee x_3), \ldots, (\neg x_k \vee x_1)$.
- Now ϕ is satisfiable iff the modified expression ϕ' is satisfiable as for each x appearing $k > 3$ times in ϕ, the truth values of $x_1, \ldots, x_k$ are the same for each truth assignment.

SAT—cont'd

- The boundary between **P** and **NP**-completeness is between 2SAT and 3SAT.
- Polynomial algorithm for 2SAT using graph reachability.
- Let ϕ be an instance of 2SAT. Define a graph $G(\phi)$ as follows: The vertices of $G(\phi)$ are the variables of ϕ and their negations. 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 the literal 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 $G(\phi)$.

SAT—cont'd

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

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

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

- MAX2SAT generalizes 2SAT by asking whether there is a truth assignments which satisfies at least K clauses for a given integer K (but not necessarily all clauses as in 2SAT).

Theorem. MAX2SAT is **NP**-complete.

(See tutorials.)

SAT—cont'd

NAESAT is a subclass of 3SAT such that there is a truth assignment where in no clause all three literals equal in truth value.

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

Proof.

- CIRCUIT SAT was shown to be **NP**-complete and a reduction from CIRCUIT SAT to SAT has been presented.
- Consider now the clauses in the reduction and add to all one- and two-literal clauses among them the same literal, say z, to make the clauses to have 3 literals.
- Claim: the resulting set of clauses is in NAESAT iff the original circuit is satisfiable.

($\Rightarrow$) Suppose that a truth assignment T satisfies all clauses (in the sense of NAESAT). Then the complementary truth assignment $\overline{T}$ satisfies the clauses in the sense of NAESAT, too.
In one of these z is **false** and, hence, the original set of clauses is satisfiable implying that the circuit is satisfiable, as well.
($\Leftarrow$) Suppose a truth assignment satisfying the circuit.
Then there is a truth assignment T satisfying the clauses in the original reduction to SAT. Take $T(z) =$ **false**. Now in no clause all literals are **true** (they cannot be all **false**): (i) clauses for **true**/**false**/NOT/variable gates contain z which 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) OR gates behave similarly. $\square$

Graph-Theoretic Problems

- Consider undirected graphs.
- INDEPENDENT SET:
  INSTANCE: An undirected graph $G = (V, E)$ and an integer K.
  QUESTION: Is there an independent set $I \subseteq V$ with $|I| = K$.
  (A set $I \subseteq V$ is independent if $i, j \in I$ implies that there is no edge between i and j).

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

By examining the subclass of graphs needed in the reduction from SAT to INDEPENDENT SET:
Corollary. 4-DEGREE INDEPENDENT SET is **NP**-complete.

Graph problems—cont'd

General pattern: problems in graph theory closely related.

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 if $i, j \in I$ implies $[i, j] \in E$)

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 if $[i, j] \in E$, then $i \in C$ or $j \in C$.

Graph problems—cont'd

- INDEPENDENT SET can be reduced to CLIQUE: an independent set of a graph is a clique of the *complement* graph.
  Hence, G, K is reduced to $\overline{G}, K$.
- INDEPENDENT SET can be reduced to NODE COVER: I is an independent set of a graph iff $V - I$ is a node cover the graph.
  Hence, G, K is reduced to $G, |V| - K$.

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

Graph problems—cont'd

- 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 $(S,V - S)$ is the number of edges between S and $V - S$.
- MIN CUT (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.
- But maximizing is harder:

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

Proof.

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

- For a set of clauses $C_1, \dots, C_m$ we construct a graph $G = (V,E)$ that has a cut of size 5m iff the clauses are satisfied.
- The nodes are $x_1, \dots, x_n, \neg x_1, \dots, \neg x_n$ where $x_1, \dots, x_n$ are the variables in the clauses.
- 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.

(Proof–cont'd)

- Suppose there is a cut $(S,V - S)$ of size $5m$ or more.
- Then it can be assumed that all variables are separated 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., the clauses are NAESAT satisfied.
- Conversely, a satisfying truth assignment gives a cut of size $5m$.

Graph problems—cont'd

- In applications of graph partitioning, the sizes of S and $V - S$ are often constrained.

  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.
$G = (V,E)$ 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 $|S| = |V - S|$).

Graph problems—cont'd

- Minimization problem (MIN CUT + 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 of G has a bisection of size $n^2 - K$ or less. □

Establishing NP-completeness

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

- Work on examples.
- Look at known **NP**-complete problems.
- Design a reduction

  From *a known* **NP***-complete problem* **to** Q
- Typical ingredients in a reduction

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

Graph problems—cont'd

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 gadget* for each variable x_i (a truth assignment).
- *Consistency gadgets* (XOR) enforce that all occurrences of x_i have the same truth value and $\neg x_i$ the opposite.
- *Constraint gadgets* guarantee that all clauses are satisfied.

(Proof—cont'd)

- Constructing $R(\phi)$:
  - *Choice gadget* for each variable x_i connected in series
  - A *constraint gadget* (triangle) for each clause with an edge assigned to 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 are traversed and there is no Hamilton path.

(Proof—cont'd)

- If there is a Hamilton path, ϕ 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 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 path can be found leading to 3 and then to 2. □

TSP

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 such that there is a tour of length B or less iff G has a Hamilton path.
Distance matrix: there are n cities and distance $d_{ij} = 1$ if there is an edge $[i, j]$ in G and 2 otherwise; 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. □

Coloring problems

k-COLORING:
INSTANCE: A graph.
QUESTION: Is there an assignment of one of k colors to each of the nodes of the graph such that any two nodes connected by an edge do not have the same color.

☞ Two colors is easy (in **P**) but already three is not.

Coloring problems–cont'd

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

- Reduction from NAESAT: For a set of clauses $C_1, \ldots, C_m$ involving variables $x_1, \ldots, x_n$ construct a graph G such that G can be colored with colors $\{0, 1, 2\}$ iff there is a truth assignment with no clause in which all literals have the same truth value.

- *Choice gadget*: a triangle
  For each variable x_i, a triangle $[a, x_1, \neg x_i]$, 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 l where l is the jth literal in C_i.

(Proof—cont'd)

➤ Claim: G can be colored with $\{0,1,2\}$ iff there is a truth assignment such that the clauses are NAESAT satisfiable.

➤ ($\Rightarrow$) 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 $T(x_i) = \mathbf{false}$.

Assume that the coloring yields true to all literals in a clause C_i. Then color 1 cannot be used for coloring $[C_{i1}, C_{i2}, C_{i3}]$ and there is no coloring. Similarly, if the coloring yields false to all literals in C_i. Then color 0 cannot be used for coloring $[C_{i1}, C_{i2}, C_{i3}]$ and there is no coloring. Hence, the clauses are NAESAT satisfiable.

(Proof—cont'd)

➤ ($\Leftarrow$) Let T be a NAESAT satisfiable truth assignment. Then a coloring of G is obtained by

- setting color 2 to a,
- color 1 to x_i and 0 to $\neg x_i$ if $T(x_i) = \mathbf{true}$ else vice versa
- coloring the triangles $[C_{i1}, C_{i2}, C_{i3}]$ as follows:
  Pick two literals with opposite truth values and color them with
  $\{0,1\}$: 0 if the literal is **true** and 1 if the literal is **false**;
  Color the third with 2. □

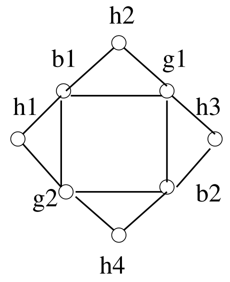
Sets and numbers

TRIPARTITE MATCHING:
INSTANCE: Sets B, G, H each containing n elements and a ternary relation $T \subseteq B \times G \times H$
QUESTION: Is there a set of n triplets in T no two of which have a component in common.

Theorem. TRIPARTITE MATCHING is **NP**-complete.
Proof. By a reduction from 3SAT where each variable 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 triplets (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).

(Proof—cont'd)

➤ The combined 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. (Notice that k can be assumed to be 1 or 2). The case $k = 2$ 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} and h_{2i-1}

(Proof—cont'd)

- Note that "$T(x) = \textbf{true}$" matching leaves the homes for x unoccupied and "$T(x) = \textbf{false}$" those for $\neg x$ unoccupied.
- Now the idea is that 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 the clause.
- 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 $|H|$ triplets with the ith boy and every home.
- Now a tripartite matching exists iff the set of clauses is satisfiable.

Other set problems

- 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.
- 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.
- 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.

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

Proof. (By generalization)

- TRIPARTITE MATCHING can be reduced to EXACT COVER BY 3-SETS by noticing that it is a special case where U is partitioned in 3 equal sets B, G, H and $F = \{\{b, g, h\} \mid b \in B, g \in G, h \in H\}$.
- 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 $K = m$. □

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 (where $x_i = 1$ if S_i in the cover and otherwise $x_i = 0$):

$$0 \leq x_1 \leq 1, \ldots, 0 \leq x_n \leq 1 \qquad a_{11}x_1 + \cdots + a_{1n}x_n \geq 1$$

$$\Sigma_{i=1}^{n} x_i \leq B \qquad \vdots$$

$$a_{m1}x_1 + \cdots + a_{mn}x_n \geq 1$$

where $a_{ij} = 1$ if ith element of U is in the set S_j, otherwise $a_{ij} = 0$.

Sets and numbers—cont'd

- LINEAR PROGRAMMING (INTEGER PROGRAMMING where non-integer solutions are allowed) is in **P**.
- 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.

Sets and numbers—cont'd

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

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

This is done by a reduction from EXACT COVER BY 3-SETS as follows.

- Reduction from EXACT COVER BY 3-SETS where $U = \{1, 2, \ldots, 3m\}$ and the sets S_i are given as bit vectors $\{0,1\}^{3m}$ and $K = 2^{3m} - 1$, i.e., find a subset of bit vectors that sum to K.
- This does not quite work because of the carry bit.
- A correction:
  use base $n+1$ instead of 2.
- Then S_i corresponds to
  $w_i = \Sigma_{j \in S} (n+1)^{3m-j}$ and
  $K = \Sigma_{j=0}^{3m-1} (n+1)^j$

$$
\begin{array}{cccccc}
\rightarrow & 0 & 1 & \ldots & 0 & 0 \\
\rightarrow & 1 & 0 & \ldots & 0 & 0 \\
 & \vdots & & & & \\
\rightarrow & 0 & 0 & \ldots & 1 & 1 \\
\hline
 & 1 & 1 & \ldots & 1 & 1
\end{array}
$$

Pseudopolynomial algorithms and strong NP-completeness

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 $V(w,i)$ $(w = 1, \ldots, W,\ i = 1, \ldots, n)$ 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 of KNAPSACK is answered "yes" iff there is an entry $V(w,i) \geq K$. □

Strong NP-completeness

- Do we have $\mathbf{P} = \mathbf{NP}$ established now?
- The 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 pseodopolynomial.
- 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 $p(n)$, a polynomial.

  ☞ Strongly **NP**-complete problems cannot have pseodopolynomial algorithms (unless $\mathbf{P} = \mathbf{NP}$).
- Strongly **NP**-complete problems: SAT, MAX CUT, TSP(D), HAMILTON PATH,... (Exception: KNAPSACK).

Strong NP-completeness—cont'd

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