SUBSET-SUM -> PARTITION

Let (L, B) be an instance of subset sum, where L is a list (multiset) of numbers, and B is the target sum. Let $S = \sum L$. Let L' be the list formed by adding $S + B, 2S - B$ to L. Observe that $\sum L' = 4S$

($\Longrightarrow$) If there is a sublist $M \subseteq L$ summing to B, then L' can be partitioned into two equal parts: $M \cup \{2S - B\}$ and $L \setminus M \cup \{S + B\}$. Indeed, the first part sums to $B + (2S - B) = 2S$, and the second to $(S - B) + (S + B) = 2S$.

($\Longleftarrow$) If L' can be partitioned into two equal parts P_1, P_2, then we will prove there is a sublist of L summing to B.

Recall that since $\sum L' = 4S$ then $\sum P_1 = \sum P_2 = 2S$.

Note that $(S + B) + (2S - B) = 3S \neq 2S$ therefore these two elements reside in opposite sides of the partition.

Without loss of generality, suppose that $2S - B \in P_1$. Therefore the rest of P_1's must come exactly from L and must sum to B which, so that we've found a sublist which sums to B, as needed.

MAX-CUT -> MAX-2-SAT

The reduction can be done in the following way.

Given an instance $G = (V, E)$ for Max-Cut, we create an instance of problem B with one variable x_v per vertex $v \in V$, and two constraints per edge $(v, w) \in E$:

$$(x_v + x_w) > 0$$
$$(x_v + x_w) < 2$$

Every subset of V corresponds to a binary assignment for the x_v in a natural way. Furthermore, the number of constraints satisfied is exactly $|E| + |E(U, V)|$, where U, V are the 2 sets after the cut, and a variable x_v is assigned 1 iff it's assigned to U.
Set $j = k + |E|$.

Intuitively, the edge (v, w) is cut iff $(x_v + x_w) > 0$, $(x_v + x_w) < 2$ are both satisfied and this is why this reduction works.

The reduction takes in $|V|$ vertices and $|E|$ edges, then transforms them directly to $|V|$ variables and $2|E|$ constraints, so the reduction takes $O(|V| + |E|)$ time, hence is a polynomial time reduction.

YES-instance of A iff YES-instance of B: Notice that for a pair of constraints, $(x_v + x_w) > 0$, $(x_v + x_w) < 2$, there will be exactly one satisfied if both vertices are in the same set after the partition, and both satisfied if the 2 vertices are in distinct sets. Due to the construction, there will be $|E|$ such pairs, and hence the number of clauses being satisfied will be $|E| + |E(V_0, V_1)|$, which is exactly saying $j = k + |E|$.

HAM-CYCLE -> TSP

Remark: Problem A is TSP (decision version), and Problem B is HAMILTON-CYCLE.

Denote $B(G)$ as the instance of B with graph G and $A(G, k)$ be the instance of A with graph G and threshold k. From a $B(G)$ instance, we convert to A instance as follows.

1. Construct a complete weighted graph $G' = (V', E')$ such that $V' = V$ and the weight of $e \in E'$ is 1 if $e \in E$ and N otherwise (pick $N > |V|^3 + 100$).
2. Run $A(G', |V|)$.

First, this reduction is polynomial time, since step 1 takes $\approx O(V^2)$ time (we need to check all edges and vertices, which is at most $O(V^2)$ and this is polynomial with respect to our input of $O(V + E) = O(V^2)$ edges and vertices because $|E| \in O(V^2)$).

We then prove that $A(G', |V|)$ is a YES-instance iff $B(G)$ is a YES-instance. For the if part, because $A(G', |V|)$ is a YES-instance, there is a cycle in A that only have edges with weight 1 (due to our graph construction, as $N > |V|$). This means that all edges in this cycle are in B and this is a cycle. Hence, $B(G)$ is a YES-instance. For the only if part, suppose $C \subset E$ is a cycle. Then, the weight of this cycle in G' is $|V|$, which means that $A(G', |V|)$ is a YES-instance. We are done.

VC -> IS

vertex cover (VC)

- given an undirected graph $G = (V, E)$, $X \subseteq V$ is a vertex cover if $\forall u \in E, (\forall v \in E, ((u \in X \vee v \in X) \wedge \sim (u \in X \wedge v \in X)))$
- optimization: compute VC of smallest size
- decision: does there exist a VC of size $\leq k$

independent set (IS)

- given an undirected graph $G = (V, E)$, $X \subseteq V$ is an independent set if $\forall u \in X, (\forall v \in X, ((u, v) \notin E))$
- optimization: compute the IS of largest size
- decision: does there exist an IS of size $\geq k$

📌 show that $X \subseteq V$ is a vertex cover of G iff $V \setminus X$ is an independent set of G

- (→) $X \in VC \Rightarrow (V \setminus X) \in IS$
  - suppose $X \in VC, \forall u \in E, (\forall v \in E, ((u \in X \vee v \in X))$
  - let $Y = V \setminus X$
  - $\forall u \in Y, (\forall v \in Y, ((u, v) \notin E))$ by definition of IS
  - proof by contradiction: suppose $(u, v) \in E$, then $X \notin VC$ by definition of VC
  - therefore, $Y \in IS$
- (←) $(V \setminus X) \in IS \Rightarrow X \in VC$
  - let $Y = V \setminus X$
  - suppose $Y \in IS, \forall u \in Y, (\forall v \in Y, ((u, v) \notin E))$ by definition of IS
  - at most 1 of $\{u, v\} \in X$, therefore, at least 1 of $\{u, v\} \in Y$ which means edge (u, v) is covered by Y
  - therefore, $X \in VC$

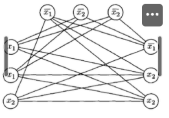
📌 show that the reduction f takes poly-time

- given input (G, k), f returns $(G, n - k)$ which means it takes poly-time

3-SAT -> MAX-CLIQUE

- given Φ with m clauses over n variables
- nodes of G organized into k groups of 3, each corresponding to a clause C_i and each node corresponds to a literal in C_i
- edges of G connect all nodes but two types of pairs (*exceptions*):
  1. nodes in the same triple
  2. nodes that are complementary

$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2).$$



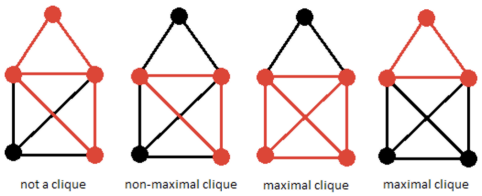
- proof: boolean formula is satisfiable iff G has a k-clique
  - (→) boolean formula is satisfiable implies G has a k-clique
    - suppose the boolean formula is satisfiable
    - at least 1 literal is true in every clause → in each C_i in the graph, select 1 node corresponding to a true literal in the assignment
      - if more than 1 literal can be chosen, choose arbitrarily
    - nodes selected form a k-clique
    - each pair of nodes is joined by an edge as no pair fits one of the *exceptions*, they cannot be from the same clause, and they cannot have contradictory labels (i.e. x_i and $\bar{x}_i$)
    - therefore, G has a k-clique
  - (←) G has a k-clique implies boolean formula is satisfiable
    - suppose G has a k-clique
    - no two of the clique's nodes occur in the same clause since the nodes of the same clause aren't connected by edges
    - each of the clauses contains exactly one of the k-clique nodes
    - assign truth values to variables such that each literal in a clique node is true
    - assignment satisfies Φ since each clause has a clique node and each clause contains a literal that is assigned true
    - therefore, Φ is satisfiable

VC -> HITTING-SET

- hitting set: for a set of sets $\{S_1, S_2, \ldots, S_n\}$, set H is a hitting set if $\forall S_i \in S, H \cap S_i \neq \emptyset$ and the hitting set problem states that given the set of sets and k, decide if there exists a hitting set of size at most k
- hitting set is in NP: hitting set of size at most k is the certificate, size of certificate is polynomial, verification is also polynomial
- transformation: given an instance $(G, k) \in VC$, $\forall e \in E, S_e = \{u, v\}$ (create a set for every edge)
  - instance of hitting set: $\{S_e | e \in E\}, k$
- proof: $X \subseteq V$ is a VC of G of size at most k iff $\{S_e | e \in E\}, k$ is a hitting set of at most k
  - ($\rightarrow$) $X \subseteq V \in VC \Rightarrow \{S_e | e \in E\}, k$ is a hitting set of at most k
    - suppose X is a VC of G
    - $\forall u \in V, (\forall v \in V, ((u \in X \vee v \in X) \wedge \sim (u \in X \wedge v \in X)))$ by definition of VC
    - $\forall (u, v) \in E$, at most 1 vertex is found in X by definition of VC
    - $\therefore$ every $x \in X$ is a vertex from each set of edges in the hitting set transformation
    - every vertex is an element that would appear in the hitting set
    - given that X is at most k size, then there are at most k elements in the hitting set, assuming that there are no duplicates
      - if there are duplicates, then the hitting set has less than k elements
  - ($\leftarrow$) $\{S_e | e \in E\}, k$ is a hitting set of at most k size $\Rightarrow X \subseteq V \in VC$
    - suppose $\{S_e | e \in E\}, k$ is a hitting set of at most k size
    - notice that the elements in the hitting set correspond to the nodes in the VC

IS -> MAX-CLIQUE

- given an undirected graph G and an integer k, is there a clique of size at least k or not in G?
  - a **clique**, C, in an undirected graph $G = (V, E)$ is a subset of the vertices, $C \subseteq V$, such that every two distinct vertices are adjacent
    - everything is one-edge away from each other



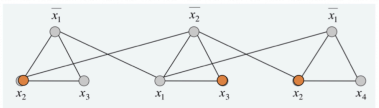
- in layman terms, an IS is one where every vertex in the set does not have an edge between them; a clique is one where every vertex in the set has an edge between them
- therefore, an IS is a clique in the complement of the graph G
- given (G, k), create (G^C, k)
  - G has an independent set of size k iff G^C has a clique of size k

3-SAT -> IS

- given an instance Φ of 3-SAT, construct an instance of (G, k) of IS such that G has an independent set iff Φ is satisfiable

> 📌 **biggest property of n-SAT**
>
> if an edge is drawn between a literal and its negation, then only one vertex of that edge can be chosen at a time

- reduction: let G have 3 vertices per clause, with 1 per literal
  - connect 3 literals in a clause in a triangle
  - connect each literal to each of its negations
  - k = number of clauses
- ($\leftarrow$) suppose Φ is a YES-instance
  - take any satisfying assignment for Φ and select a true literal from each clause, these k vertices form an IS of G



$$(\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$$

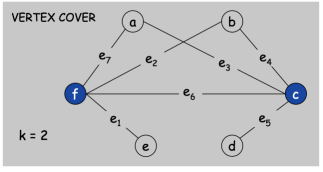
- ($\rightarrow$) suppose (G, k) is a YES-instance
  - let S be the IS of size k
  - each of the k triangles must contain exactly one vertex in S and use these literals as true

> 📌 3-SAT $\leq_P$ IS $\leq_P$ VC
> IS $\leq_P$ Max-Clique

VC -> SET-COVER

Claim. VERTEX-COVER $\leq_P$ SET-COVER.
Pf. Given a VERTEX-COVER instance G = (V, E), k, we construct a set cover instance whose size equals the size of the vertex cover instance.

Construction.
- Create SET-COVER instance:
  - k = k, U = E, S_v = {e ∈ E : e incident to v }
- Set-cover of size ≤ k iff vertex cover of size ≤ k. ▪



SET COVER

U = { 1, 2, 3, 4, 5, 6, 7 }
k = 2
S_a = {3, 7}
S_b = {2, 4}
S_c = {3, 4, 5, 6}
S_d = {5}
S_e = {1}
S_f = {1, 2, 6, 7}

A list of useful NP-Complete problems

1 Satisfiability

Circuit Satisfiability

Instance: A circuit C with m inputs
Question: Is there an input for C such that C returns true for it.

Definition 1.1 A boolean formula is in conjunctive normal form (***CNF***) if it is a conjunction (AND) of several *clauses*, where a clause is the disjunction (or) of several *literals*. A literal is either a variable or a negation of a variable.

SAT

Instance: A CNF formula F with n variables
Question: Is there an assignment to the variables such that F is TRUE?

Definition 1.2 ***3CNF*** formula is a CNF formula with *exactly* three literals in each clause.

S3AT

Instance: A 3CNF formula F with n variables
Question: Is there an assignment to the variables such that F is TRUE?

2 Clique/independent set/vertex cover

Definition 2.1 A ***clique*** is a complete graph, where every pair of vertices are connected by an edge.

Clique

Instance: A graph G, integer k
Question: Is there a subgraph H in G with k vertices, such that H is a clique?

Definition 2.2 A set S of nodes in a graph $G = (V, E)$ is an ***independent set***, if no pair of vertices in S are connected by an edge.

Independent Set

Instance: A graph G, integer k
Question: Is there an independent set in G of size k?

Definition 2.3 For a graph G, a set of vertices $S \subseteq V(G)$ is a ***vertex cover*** if it touches every *edge* of G.

Vertex Cover

Instance: A graph G, integer k
Question: Is there a vertex cover in G of size k?

3 Coloring

Definition 3.1 A ***coloring***, by c colors, of a graph $G = (V, E)$ is a mapping $C : V(G) \to \{1, 2, \ldots, c\}$ such that every vertex is assigned a color, such that no two vertices that share an edge are assigned the same color.

3Colorable

Instance: A graph G.
Question: Is there a coloring of G using three colors?

4 Hamiltonian paths/cycles and TSP

Definition 4.1 A ***Hamiltonian cycle*** is a cycle in the graph that visits every vertex exactly once.

Hamiltonian Cycle

Instance: A graph G.
Question: Is there a Hamiltonian cycle in G?

Hamiltonian Cycle is NP-Complete both for directed and undirected graphs.

Hamiltonian Path

Instance: A graph G.
Question: Is there a Hamiltonian path in G? Namely, is there a simple path that visits all the vertices of G exactly once.

Hamiltonian Path is NP-Complete both for directed and undirected graphs. It remains NPC even if you specify the start and end vertices of the path.

Definition 4.2 A ***traveling salesman tour*** (***TSP***), is a Hamiltonian cycle in a graph. Its price is the total price of all its edges.

TSP

Instance: $G = (V, E)$ a complete graph - n vertices, $c(e)$: Integer cost function over the edges of G, and k an integer.
Question: Is there a traveling-salesman tour with cost at most k?

TSP remains NP-Complete if the graph directed/undirected or if instead of a closed tour, one is looking for a path that visits every vertex exactly once.

5 Subset sum and partition

Subset Sum

Instance: S - set of positive integers,t: - an integer number (Target)
Question: Is there a subset $X \subseteq S$ such that $\sum_{x \in X} x = t$?

Partition

Instance: A set S of n numbers.
Question: Is there a subset $T \subseteq S$ s.t. $\sum_{t \in T} t = \sum_{s \in S \setminus T} s$?

6 Three dimensional matching and set cover

3DM

Instance: X, Y, Z sets of n elements, and T a set of triples, such that $(a, b, c) \in T \subseteq X \times Y \times Z$.
Question: Is there a subset $S \subseteq T$ of n disjoint triples, s.t. every element of $X \cup Y \cup Z$ is covered exactly once.?

SET COVER

Instance: $(U, \mathcal{F}, k)$:
U: A set of n elements
$\mathcal{F}$: A family of subsets of U, s.t. $\bigcup_{X \in \mathcal{F}} X = U$.
k: A positive integer.
Question: Are there k sets $S_1, \ldots, S_k \in \mathcal{F}$ that cover U. Formally, $\bigcup_i S_i = U$?

An Annotated List of Selected NP-complete Problems

The standard textbook on *NP*-completeness is:

Michael Garey and David Johnson: *Computers and Intractability - A Guide to the Theory of NP-completeness*; Freeman, 1979.

David Johnson also runs a column in the journal *Journal of Algorithms* (in the HCL; there is an on-line bibliography of all issues)

On the Web the following sites may be of interest:

http://www.nada.kth.se/~viggo/problemlist/compendium.html

Or trying giving `NP-complete' or `NP and complete' as a search term to

http://liinwww.ira.uka.de/searchbib/index>

(This is basically a bibliography database, but, you can click on the `on-line papers' button to list electronically readable full texts).

The Center for Discrete Mathematics and Theoretical Computer Science (DIMACS) gives links to technical papers, abstracts, and other information concerning algorithms, approximation techniques and properties of *NP*-complete problems. Other general algorithmics and complexity related sites may be found at:

The Electronic Colloquium on Computational Complexity (ECCC)

A number of relevant journals are available on-line through the University Library Web Pages (these are only accessible to members of Liverpool University). The following journals are all available and publish research papers in the general areas of Algorithmics and Complexity Theory:

- *Acta Informatica*
- *Computer Journal*
- *Information and Computation*
- *Journal of Algorithms*
- *Journal of Computer and System Sciences*
- *SIAM journal on computing*

In the list of NP-complete problems below, the form of a typical entry is as follows:

Number: 0

Name: Wombat Eating Assignment (WEA) [DU0] 2

Input: A set *W* of *n* wombats; a forest *T* of *m* eucalyptus trees (*m>=n*); a habitation mapping, *mu:W->subsets of T*, such that for each wombat, w, *mu (w)* defines the set of eucalyptus trees, *t* in *T*, in which the wombat lives.

Question: Is there a mapping, *E*, from the set of wombats to the set of trees which satisfies both:

- Every wombat is assigned a unique eucalyptus tree by E, i.e. if $E(u)=t$ then $E(v)/=t$ for all distinct pairs of wombats u and v;
- E is consistent with the habitation mapping, mu, i.e. if $E(w)=t$ then t belongs to the set of trees, $mu(w)$ in which the wombat lives?

Comments: Can be solved by efficient methods using (0,1)-network flow maximisation methods applied to bipartite graphs, i.e. it's a matching problem.

The first line gives the unique identifying number for the problem (as described above). The second line gives the name of the problem and (if there is one) its usual abbreviation. These are how the problem is usually referred to in research papers, textbooks, etc. If you are looking for material on the Web, then these should give reasonable terms to supply to search engines. The code in square brackets is a reference to the classification in Garey and Johnson's book (with the exceptions of Problem 14, 86, 87, 88).

The next part of the description indicates what a typical *instance* (i.e. input) consists of: notice, as in the example above, that this will usually consist of a number of distinct objects which must be represented in trying to solve the problem: sometimes these can be quite involved structures, such as graphs, sets, mappings, logical expressions, etc; and, sometimes quite simple forms such as a single integer. The main part of a problem definition is *always* formulated as a *question*. This is the core of the decision problem description and defines precisely what property of the input instance must be determined in solving it. Thus, in the example given, one is trying to decide if a given input instance is such that a mapping from wombats to trees, satisfying certain conditions is possible. The final part (which is not always present) gives some (not necessarily useful) comments regarding the problem.

Notice that *one* way in which the WEA problem could be solved is by *exhaustively* considering each distinct way of mapping from single wombats to single trees until either one found an assignment that met the required conditions given in the Question; or one had no further mappings left to consider. Clearly the former case would lead to the answer **true** being returned; the latter to the answer **false** being given.

Selected NP-Complete Problems

Number: 1

Name: 3-Satisfiability (3-SAT) [LO2] 2

Input: A set of *m clauses - C_1 ,C_2,...,C_m -* over a set of *n* Boolean valued variables X_n=< x_1,x_2,...,x_n>, such that each clause depends on exactly three distinct variables from X_n. A **clause** being a Boolean expression of the form y_i+y_j+y_k where each *y* is of the form *x* or *-x* (i.e. negation of *x*) with *x* being some variable in X_n. For example if *n=4* and *m=3* a possible instance could be the (set of) Boolean expressions: C_1=(x_1+(-x_2)+(-x_3)); C_2=(x_2+x_3+(-x_4)); C_3=((-x_1)+x_3+x_4);

Question: Can each variable x_i of X_n be assigned a Boolean value $alpha_i$ in such a way that every clause evaluates to the Boolean result **true** under the assignment < x_i:=$alpha_i$;*1*<=*i*<=*n*>?

In the example instance, the assignment < x_1:=***true***;x_2:=***false***;x_3:=***true***; x_4:=***false***> is such that each of the three clauses takes the value **true**.

Comments: For reasons that will become clearer at the end of the course, this problem had to be Number 1. Some relevant material concerning this problem might be found in the special issue (Volume 81, 1996) of the

journal *Artificial Intelligence* that is available in the Library.

Number: 2

Name: Graph 3-Colourability (3-COL) [GT4] 1

Input: An *n*-node undirected graph *G(V,E)* with node set *V* and edge set *E*.

Question: Can each node of *G(V,E)* be assigned exactly one of three colours - Red, Blue, Green - in such a way that no two nodes which are joined by an edge, are assigned the same colour?

Number: 3

Name: Monochromatic triangle [GT6] 2

Input: An *n*-node undirected graph *G(V,E)* with node set *V* and edge set *E*.

Question: Can the edges, *E*, of *G* be partitioned into two disjoint sets E_1 and E_2, in such a way that neither of the two graphs G_1*(V,*E_1*)* or G_2*(V,*E_2*)* contains a triangle, i.e. a set of three distinct nodes *u,v,w* such that *{u,v}*, *{u,w}*, *{v,w}* are all edges?

Number: 4

Name: Clique [GT19] 1

Input: An *n*-node undirected graph *G(V,E)* with node set *V* and edge set *E*; a positive integer *k* with *k*<=*n*.

Question: Does *G* contain a *k*-clique, i.e. a subset *W* of the nodes *V* such that *W* has size *k* and for each distinct pair of nodes *u*, *v* in *W*, *{u,v}* is an edge of *G*?

Number: 5

Name: Bipartite Subgraph [GT25] 2

Input: An *n*-node undirected graph *G(V,E)* with node set *V* and edge set *E*; a positive integer *k* with *k*<=*|E|*.

Question: Is there a subset, *F* of the edges of *G*, having size at least *k* and such that the graph *H(V,F)* is bipartite?

Comments: *G(V,E)* is *bipartite* if the nodes can be partitioned into two disjoint sets *U* and *W* such that every edge of *G* connects a node in *U* to a node in *W*, i.e. no two nodes in *U* (resp. *W*) form an edge of *G*.

Number: 6

Name: Vertex Cover [GT1] 1

Input: An *n*-node undirected graph *G(V,E)* with node set *V* and edge set *E*; a positive integer *k* with *k*<=*n*.

Question: Is there a subset *W* of *V* having size at most *k* and such that for every edge *{u,v}* in *E* at least one of *u* and *v* belongs to *W*?

Number: 7

Name: Chromatic Index (Edge Colouring) [OPEN5] 3

Input: An *n*-node undirected graph *G(V,E)* with node set *V* and edge set *E*; a positive integer *k* with *k*<=*|E|*.

Question: Can the *edges* of *G* be assigned exactly one of *k* colours in such a way that no two edges which have a common node as an endpoint are assigned the same colour?

Comments: Vizing's Theorem from Graph Theory shows that the only `hard' case for this problem is when *k* is equal to the maximum degree of *G*. The degree of a node, *v*, is the number of edges in the graph with *v* as an end-point; the degree of a graph is the maximum degree of any node in the graph. Chromatic Index was shown to be *NP*-complete soon after the 1979 edition of Garey and Johnson went to press, hence the OPEN categorisation.

The classical paper on sequential methods is:

- A.M. Gibbons and O.A. Ogunyode: Optimal edge-colouring of almost all simple graphs in polynomial time. *Random Graphs `85: Conf. Record of 2nd International Seminar on Random Graphs and Probabilistic Methods in Combinatorics, Poznan*, North-Holland, (1985).

There are also a number of methods presenting efficient *parallel* algorithms for various special types of graph. Some of the ideas presented in these may be useful in formulating sequential techniques, e.g.

- A.M. Gibbons and W. Rytter: Optimally edge-colouring outerplanar graphs is in *NC*. *Theoretical Computer Science*, 71, (1990), pp. 401-411
- A.M. Gibbons and W. Rytter: Fast parallel algorithms for optimal edge-colouring of some tree-structured graphs. *Fundamentals of Computation Theory (FCT) `87*, Springer-Verlag, 1987

Number: 8

Name: Multiprocessor Scheduling [SS8] 4

Input: A set, *T*, of *tasks* and for each task *t* in *T* a positive (integer) running time *len(t)*. A positive integer *D*, called the *deadline*.

Question: Is there a 2-processor schedule for the tasks that completes within the deadline, *D*, i.e. is there a function *sigma:T->**N*** such that both of the following hold:

- For all *u>=0* the number of tasks *t* in *T* for which *sigma(t)<=u< sigma(t)+len(t)* is at most 2.
- For all tasks *t*, *sigma(t)+len(t)<=D*?

Comments: This is the simplest avatar of a very large number of *NP*-complete processor scheduling problems and, unsurprisingly given its practical applications in multiprogramming Operating Systems on small parallel systems, there is an enormous range of literature on approximation and heuristic techniques to tackle it, see e.g. relevant sections of this bibliography. The formal framework given by (a) and (b) can be interpreted as meaning: a) at any given time at most two tasks are being executed; b) every task has been completed by the deadline *D*.

Number: 9

Name: Comparative Divisibility [AN4] 3

Input: A (strictly increasing) sequence *A=<* a_1,a_2,...,a_n> and a (strictly increasing) sequence *B=<* b_1,b_2,...,b_m> of positive integers.

Question: Is there an integer, *c*, such that *Divides(c,A)> Divides(c,B)*, where *Divides(x,Y)* (*Y* being a sequence of positive integers) is the number of elements, *y* in *Y*, for which *x* is an exact divisor of *y*?

Comments: You may think that this has an obvious fast algorithm, and, indeed the algorithm in question is obvious: what it is not is *efficient*. Consider: how many *bits* are needed to store the input data (assuming, without loss of generality, that a_n>=b_m)? How many steps, however, is this `obvious method' taking in the worst-case? It is important to realise that representing integer values in *unary* is not considered to be a `reasonable' approach (the number 2^{50}-*1* requires 2^{50} digits in unary but only *50* digits in binary).

Number: 10

Name: Cyclic Ordering [MS2] 3

Input: A finite set, *A*, and a collection, *C*, of *ordered triples (a,b,c)* of *distinct* elements from *A*.

Question: Is there a one-to-one mapping *f:A->{1,2,3,...,|A|}* (i.e. a function which maps each element of *A* to a number between *1* and |*A*| with no two distinct elements of *A* being mapped to the same number) such that for each *(a,b,c)* in *C* one of

f(a) < f(b) < f(c) ; f(b) < f(c) < f(a) ; f(c) < f(a) < f(b)

holds?

Comments: For any triple *(a,b,c)* there are 6 (six) possible orderings that could result for *((a),f(b),f(c))*. The question being asked is whether there is a choice of function that forces every specified triple into one of three `legal' orderings. Garey and Johnson has a typographical error in describing this problem, whereby each triple `in *A*' is referred to. Obviously `in *C*' is intended.

Number: 11

Name: Quadratic Diophantine Equations [AN8] 3

Input: Positive integers *a*, *b*, and *c*.

Question: Are there two positive integers *x* and *y* such that *(a*x*x)+(b*y)=c*?

Comments: The comments regarding Problem 9 (Comparative Divisibility) are also pertinent with respect to this problem. Again, there is an `obvious' algorithm that, on the surface, appears to be efficient and is seen not to be so only once one compares the *input size* (space needed to represent the input data) to the actual computation time in the worst-case.

Number: 12

Name: Maximum 2-Satisfiability [LO5] 3

Input: A set of m clauses $C_1,C_2,...,C_m$ over n Boolean valued variables X_n, where each clause depends on *two* distinct variables from X_n; a positive integer k with $k<=m$.

Question: Is there an assignment of Boolean values to the variables X_n such that at least k distinct clauses take the value **true** under the assignment?

Comments: For relevant definitions see under Problem 1. *2-SAT*, a natural variant of the problem *3-SAT* described in Problem 1, can be solved in $O(m)$ steps. The simple variation described here (whereby one asks whether at least some number of clauses can be made **true** simultaneously) is much more difficult.

Number: 13

Name: Register Sufficiency [PO1] 5

Input: A *directed*, *acyclic* graph $G(V,A)$ in which each node has at most two out-going edges; a positive integer k.

Question: Is there a k or fewer register `computation' for G, i.e. is there an ordering $< v_1,v_2,...,v_n>$ of the (n) nodes of G and a sequence $< S_0;S_1;...;S_n>$ of subsets of V which satisfy all of the following:

- For all i, S_i contains at most k nodes from V.
- S_0 contains no nodes; S_n contains all of the nodes in V with no in-coming edges in G (recall that G is *directed*).
- For each i, with $1<=i<=n$: v_i is in S_i; S_i-$\{v_i\}$ is a subset of the nodes in S_{i-1}; and S_{i-1} contains all nodes u for which (v_i,u) is an edge in A, i.e. all nodes, u, for which there is an edge **from** v_i **to** u?

Comments: The background to this problem comes from developing Code Generation methods in compilers for High-Level Languages. On early architectures, performing arithmetic operations with both operands stored in registers was significantly faster than fetching an operand from main memory. Such machines, however, had only a small (16-32) number of fast registers. Thus in generating assembly code to represent the computation of a lengthy arithmetic expression, it was necessary to use registers efficiently. A common first stage in compiling such expressions was to represent the computation as a (so-called) `straight-line program' which in turn could be modelled as a directed acyclic graph. The object of the Register Sufficiency Problem is to determine if such a `straight-line program' can be evaluated using only the specified number of available registers.

Number: 14

Name: Central Slice of Half-Clique 2

Input: An $2n$-node undirected graph $G(V,E)$ with node set V and edge set E.

Question: Does *either* of the following hold true of $G(V,E)$

- G contains *at least* $(2n^2-n)/2+1$ edges.
- G contains *exactly* $(2n^2-n)/2$ edges and G contains an n-clique?

Comments: For the definition of `k-clique' see Problem 4 above. This specific variant of the CLIQUE problem post-dates Garey and Johnson's 1979 text by 5 years. Its classification as *NP*-complete is, originally,

proved in:

- P.E. Dunne: Techniques for the analysis of monotone Boolean networks; Ph.D. dissertation, Univ. of Warwick, October 1984; (Theory of Computation Report No. 69, Dept. of Comp. Sci., Univ. of Warwick, 1984).

The published version appears in

- P.E. Dunne: The Complexity of Central Slice Functions. *Theoretical Computer Science*, 44, (1986), pp. 247-257

For problems whose inputs are encoded by N bits, there are a number of other *NP*-complete problems that remain so even when the only `non-trivial' inputs are those with exactly $N/2$ input bits equal to *1*. Three further examples are given in the references cited.

Number: 15

Name: Decision Tree [MS15] 5

Input: A Boolean logic function, f, of n variables, X_n, described by its *1*-points, i.e. the set of assignments, $alpha=< a_1,...,a_n>$ such that $f(a_1,...,a_n)=1$; a positive integer K.

Question: Is there a *decision tree* for f that has *average path length* at most K?

Comments: A decision tree is a binary tree in which each non-leaf node is labelled with a variable from X_n, each leaf node is labelled either *0* or *1*, the edge from a non-leaf node to its left child is labelled *0*, that to its right child is labelled *1*. For a given assignment of Boolean values to the variables X_n a path can be traced starting from the root of the decision tree and following the edges marked with the value of the variable labelling each node encountered. The path terminates at a leaf node whose associated label gives the value of the function. The decision tree computes a Boolean function $f(X_n)$ if for every assignment, *alpha*, to the variables the path traced from the root under *alpha* terminates in a leaf labelled $f(alpha)$. The *average path length* of a binary tree with t leaf nodes and root v is: $sum\ from\ \{w:w\ is\ a\ leaf\}\ |Path\ from\ x\ to\ w|/t$.

This problem is a simpler (but still *NP*-complete) version of the form given in Garey and Johnson. For relevant variations and potential heuristic approaches, the papers

- P.E. Dunne and P.H. Leng, "An algorithm for optimising signal selection in demand-driven circuit simulation", *Transactions of the Society for Computer Simulation*, vol. 8, no.4, pp. 269-280, 1992
- P.E. Dunne, C.J. Gittings, and P.H. Leng, "Multiprocessor simulation strategies with optimal speed-up", *Inf. Proc. Letters*, vol. 54, no. 1, pp. 23-33, April 1995
- P.E. Dunne, P.H. Leng, and G.F. Nwana, "On the complexity of Boolean functions computed by lazy oracles", *IEEE. Trans. Comput.*, vol. 44, no. 4, pp. 495-502, April 1995

may provide some ideas of use.

Number: 16

Name: Shortest Common Superstring [SR9] 3

Input: A finite set $R=\{r_1,r_2,...,r_m\}$ of *binary strings* (sequences of *0* and *1*); positive integer k.

Question: Is there a binary string w of length at most k such that every string in R is a substring of w, i.e. for each r in R, w can be decomposed as $w=w_0rw_1$ where w_0, w_1 are (possibly empty) binary strings?

Comments: General problem allows more than two symbols (i.e. not just binary), but this simpler version remains *NP*-complete.

Number: 17

Name: Longest Circuit [ND28] 2

Input: n-node undirected graph $G(V,E)$; positive integer k.

Question: Does G contain a simple cycle containing at least k nodes?

Comments: Special cases of this are the famous Travelling Salesman Problem and Hamiltonian Circuit Problem. The latter corresponds to the cases $k=n$; the former to the case with each graph edge being weighted and also having $k=n$.

Number: 18

Name: Kernel [GT57] 2

Input: n-node *directed* graph $G(V,A)$.

Question: Does G possess a *kernel*, i.e. a subset W of the nodes V such that no two nodes in W are joined by an edge in A and such that for each node v in V-W there is a node w in W for which (w,v) is an edge in A?

Number: 19

Name: k-Closure [GT58] 2

Input: n-node *directed* graph $G(V,A)$; positive integer $k<=n$.

Question: Is there a subset W of V having size at most k and such that for all edges (u,v) in A either u is in W or v is *not* in W.

Number: 20

Name: Bandwidth [GT40] 3

Input: n-node undirected graph $G(V,E)$; positive integer $k<=n$.

Question: Is there a linear ordering of V with bandwidth at most k, i.e. a one-to-one function $f:V->\{1,2,...,n\}$ such that for all edges $\{u,v\}$ in G, $|f(u)-f(v)|<=k$?

Comments: If you need to be reminded of what a `one-to-one' function is, see Problem 10.

Number: 21

Name: Maximum Leaf Spanning Tree [ND2] 4

Input: n-node undirected graph $G(V,E)$; positive integer $k<=n$.

Question: Does G have a spanning tree in which at least k nodes have degree *1*.

Comments: A spanning tree of $G(V,E)$ is a tree $T(V,F)$ containing all the nodes of G and whose edges, F, are a subset of the edges in E. For the definition of `degree of a node' see Problem 7. The paper,

- P.E. Dunne: A result on k-valent graphs and its application to a graph embedding problem, *Acta Informatica*, 24, (1987), pp. 447-459

may be found a) in the HCL Library; b) to be completely incomprehensible; and c) to give a couple of ideas as regards the development of heurstic and approximation techniques.

Other papers of interest are given here

Number: 22

Name: Independent Set [GT20] 1

Input: n-node undirected graph $G(V,E)$; positive integer $k<=n$.

Question: Does G have an independent set of size at least k, i.e. a subset W of at least k nodes from V such that no pair of nodes in W is joined by an edge in E?

Number: 23

Name: Degree Constrained Spanning Tree [ND1] 4

Input: n-node undirected graph $G(V,E)$; positive integer $k<=n$.

Question: Does G have a spanning tree in which no node has degree greater than k?

Comments: For definitions see Problem 21 and Problem 7.

Number: 24

Name: Hamiltonian Path [GT39] 1

Input: n-node undirected graph $G(V,E)$.

Question: Is there a simple path of edges in G that contains every node in V, and thus contains exactly $n-1$ edges?

Comments: Material regarding the Hamiltonian *circuit* problem may be found to be relevant.

Number: 25

Name: Graph Partitioning [ND14] 2

Input: $2n$-node undirected graph $G(V,E)$; positive integer $k<=|E|$.

Question: Can the nodes of *G* be partitioned into 2 disjoint sets *U* and *W* each of size *n* and such that the total number of distinct edges in *E* that connect a node *u* in *U* to a node *w* in *W* is at most *k*?

Comments: There has long been interest in this problem from the area of printed circuit board (and latterly VLSI chip) design, in which areas this optimisation problem provides a natural formulation for the task of minimising the number of wires crossing between different boards (or chips). The famous approximating heuristic of Kernighan and Lin (*Bell Systems Tech. Jnl.*, 49, (1970), pp. 291-307: BSTJ can found in the HCL) appeared in this context. A survey and bibliography of algorithms for special cases is available.

Number: 26

Name: Cubic Subgraph [GT32] 3

Input: *n*-node undirected graph *G(V,E)*.

Question: Is there a (non-empty) subset *F* of the edges of *G* such that every node in the graph *H(V,F)* has either degree *3* or degree *0*.

Comments: For the definition of `degree of a node' see Problem 7.

Number: 27

Name: Travelling Salesman [ND22] 3-4

Input: A set *C* of *n cities* $\{c_1,...,c_n\}$; for each pair of cities (c_i,c_j) $(1<=i<j<=n)$ a positive integer *distance* $d_{i,j}$; a positive integer *B*.

Question: Is there an ordering $<pi(1),pi(2),...,pi(n)>$ of the *n* cities such that the value *sum from i=1 to n-1* $d_{pi(i),pi(i+1)}+d_{pi(n),pi(1)}$ is no more than *B*?

Comments: In effect what this problem is asking is whether there is a *tour* of the given collection of cities that visits each city exactly once and takes up total distance no more than *B*. There is a huge volume of literature concerning approximation methods, search heuristics, special case methods, etc for this very well studied problem. A bibliography has been compiled.

Number: 28

Name: Steiner Tree in Bipartite Graphs [ND12a] 3

Input: Bipartite graph *B(V,W,E)*; positive integer *k*.

Question: Is there a subtree of *B(V,W,E)* that includes all of the nodes of *V* and has at most *k* edges?

Comments: For the definition of bipartite graph (on two disjoint sets of nodes *V* and *W*) see Problem 5.

Number: 29

Name: Bounded Diameter Spanning Tree [ND4] 4

Input: *n*-node undirected graph *G(V,E)*; for each edge, *{u,v}* a positive integer weight *w(u,v)*; positive integer

B.

Question: Does *G* have a spanning tree *T* such that the total sum of the weights of edges in *T* is at most *B* and no simple path in *T* contains more than 5 edges?

Comments: For definitions see Problem 21.

Number: 30

Name: Optimal Linear Arrangement [GT42] 3

Input: *n*-node undirected graph *G(V,E)*; positive integer *k<=n*.

Question: Is there a one-to-one function *f:V->{1,2,...,n}* such that *sum from {{u,v} in E}* $|f(u)-f(v)|<=K$?

Comments: For definitions see Problem 10.

Number: 31

Name: Dominating Set [GT2] 2

Input: *n*-node undirected graph *G(V,E)*; positive integer *k<=n*.

Question: Does *G* contain a dominating set of size at most *k*, i.e. a subset *W* of *V* containing at most *k* nodes and such that for every node *u* in *V-W* (i.e. in *V* but *not* in *W*) there is a node *w* in *W* such that *{u,w}* is an edge of *G*?

Number: 32

Name: Path with Forbidden Pairs [GT54] 4

Input: *n*-node *directed* graph *G(V,A)*; two distinct nodes *s* and *t* belonging to *V*; finite collection $C=\{(a_1,b_1),...,(a_r,b_r)\}$ of pairs of nodes from *V*.

Question: Is there a *directed* path from the node *s* to the node *t* in *G* that contains at most one node from each pair of nodes in the collection *C*?

Comments: Several simplifications of this problem remain *NP*-complete: requiring *G* to be acyclic; requiring the collection *C* to contain only directed edges from *A* (rather than arbitrary pairs of nodes); requiring all the pairs to be disjoint.

Number: 33

Name: Oriented Diameter [GT64] 4

Input: *n*-node undirected graph *G(V,E)*; positive integer *k<=n*.

Question: Can a direction be placed on each edge *{u,v}* of *E* in such a way that the resulting *directed* graph *H(V,A)* is such that both of the following hold:

- *H(V,A)* is *strongly connected*, i.e. for each distinct pair of nodes, *v* and *w*, there exists a directed path

from *v* to *w* in *H and* a directed path from *w* to *v* in *H*.
- The *diameter* of *H(V,A)* is at most *k*, i.e. for every pair of nodes *v* and *w*, there is a path of at most *k* edges from *v* to *w* and a path of at most *k* edges from *w* to *v*?

Number: 34

Name: Rural Postman [ND27] 3-4

Input: *n*-node undirected graph *G(V,E)*; subset *F* of the edges *E*; positive integer *k<=|E|*

Question: Is there a (not necessarily simple) cycle of in *G* that contains every edge in *F* and has at most *k* edges in total?

Comments: A *simple* cycle in a graph is one in which each node visited in the cycle is visited *exactly* once. A non-simple cycle allows nodes to be visited more than once (although, obviously, each edge can only occur once). For example in the 5 node graph with edges *({1,2};{1,3};{1,4};{1,5};{2,3};{2,4};{2,5};{3,4})*, the cycle *1-> 2-> 3-> 1* is a *simple cycle*; the cycle *1-> 2-> 4-> 3-> 2-> 5-> 1* is a *non-simple* cycle since *2* is visited twice (notice that in neither case is any edge used more than once).

Number: 35

Name: Longest Path [ND29] 2

Input: *n*-node undirected graph *G(V,E)*; nodes *s* and *t* in *V*; positive integer *k*.

Question: Is there a *simple* path between *s* and *t* in *G* that contains at least *k* edges?

Comments: The similarity to Problem 17 should be noted.

Number: 36

Name: *3*-Dimensional Matching (*3DM*) [SP1] 3

Input: 3 disjoint sets *X*, *Y*, and *Z* each comprising exactly *n* elements; a set *M* of *m triples* $\{(x_i,y_i,z_i): 1<=i<=m\}$ such that x_i is in *X*, y_i in *Y*, and z_i in *Z*, i.e. *M* is a subset of *XxYxZ*.

Question: Does *M* contain a *matching*, i.e. is there a subset *Q* of *M* such that $|Q|=n$ and for all distinct pairs of triples *(u,v,w)* and *(x,y,z)* in *Q* it holds that *u/=x* and *v/=y* and *w/=z*.

Comments: The variant *2*-dimensional matching in which *2* disjoint sets *X* and *Y* form the basis of a set of pairs, can be solved by a number of fast methods.

Number: 37

Name: Set Splitting [SP4] 3

Input: A finite set *S*; A collection $C_1,...,C_m$ of subsets of *S*.

Question: Can *S* be partitioned into two disjoint subsets - S_1 and S_2 - such that for each set C_i it holds that C_i is *not* a subset of S_1 *and* C_i is *not* a subset of S_2?

Number: 38

Name: Set Packing [SP3] 3

Input: A collection $C=(C_1,...,C_m)$ of finite sets; a positive integer *k<=m*.

Question: Are there *k* sets - $D_1,...,D_k$ - from the collection *C* such that for all $1<=i<j<=k$, D_i and D_j have no common elements?

Number: 39

Name: Exact Cover by 3-Sets (*X3C*) [SP2] 3

Input: A finite set *X* containing exactly *3n* elements; a collection *C* of subsets of *X* each of which contains exactly *3* elements.

Question: Does *C* contain an *exact cover* for *X*, i.e. a sub-collection of *3*-element sets $D=(D_1,...,D_n)$ such that each element of *X* occurs in *exactly* one subset in *D*?

Number: 40

Name: Minimum Cover [SP5] 3-4

Input: A finite set *S*; A collection $C=(C_1,...,C_m)$ of subsets of *S*; a positive integer *k<=m*.

Question: Does *C* contain a *cover* for *S* comprising at most *k* subsets, i.e. a collection $D=(D_1,...,D_t)$, where *t<=k*, each D_i is a set in *C*, and such that every element in *S* belongs to at least one set in *D*?

Number: 41

Name: Partition [SP12] 3

Input: Finite set *A*; for each element *a* in *A* a positive integer *size s(a)*.

Question: Can *A* be partitioned into 2 disjoint sets A_1 and A_2 in a such a way that the sum of the sizes *s(x)* of elements *x* in A_1 is exactly the same as the sum of the sizes *s(y)* of the elements *y* in A_2.

Comments: It should be noted that it is *not* required that A_1 and A_2 contain equal numbers of elements, although even with this condition the problem is still *NP*-complete.

Number: 42

Name: Subset Sum [SP13] 3

Input: Finite set *A*; for each element *a* in *A* a positive integer *size s(a)*; a positive integer *K*.

Question: Is there a subset *B* of *A* such that the sum of the sizes, *s(x)*, of the elements *x* in *B* is *exactly* equal to *K*?

Number: 43

Name: Comparative Containment [SP10] 4

Input: A finite set X; 2 collections $R=(R_1,...,R_k)$ and $S=(S_1,...,S_m)$ each of which is a set of subsets of X; for each R_i in R, a positive integer weight $w(R_i)$; for each S_i in S a positive integer weight $w(S_i)$;

Question: Is there a subset Y of X such that: if R_Y is the set of subsets, R_i of R having Y as a subset of R_i and S_Y is the set of subsets, S_i of S having Y as a subset of S_i then the total weight of the sets in R_Y is at least the total weight of the sets in S_Y.

Number: 44

Name: Minimum Test Set [SP6] 3

Input: A finite set S; A collection $C=(C_1,...,C_m)$ of subsets of S; a positive integer $k<=m$.

Question: Is there a sub-collection $D=(D_1,...,D_t)$ of the sets in C such that: $t<=k$ and for each distinct pair of elements u, v in S there is a set $D_{u,v}$ in D that contains exactly one of u and v?

Number: 45

Name: Minimum Sum of Squares [SP19] 3-4

Input: A set A of n elements; for each element a in A a positive integer size $s(a)$; positive integers $k<=n$ and J.

Question: Can A be partitioned into k disjoint sets $A_1,...,A_k$ such that *sum from i=1 to k (sum from {x in* A_i*}* $s(x))^2<=J$?

Number: 46

Name: *3*-Partition [SP15] 3

Input: A set A of $3m$ elements; a positive integer bound B; for each element x in A a positive integer size $s(x)$ that satisfies $B/4< s(x)< B/2$, and is such that the sum of the sizes of elements in A is exactly mB.

Question: Can A be partitioned into m disjoint sets $A_1,...,A_m$ such that each A_i contains exactly 3 elements of A and each A_i has total size equal to B?

Number: 47

Name: Subset Product [SP14] 3

Input: Finite set A; for each element a in A a positive integer *size* $s(a)$; a positive integer K.

Question: Is there a subset B of A such that the *product* (i.e. result of multiplying together all) of the sizes, $s(x)$, of the elements x in B is *exactly* equal to K?

Comments: There is a subtle technical distinction between this and Problem 42: the former case has a `pseudo-efficient' algorithm obtained by allowing numbers to be represented in unary; unless all *NP*-complete problems can be solved by fast algorithms, however, the Subset Product Problem, cannot be solved by `efficient' methods using even this unreasonable input representation.

Number: 48

Name: Bin Packing [SR1] 3

Input: A finite set U of m items; for each item u in U a positive integer size $s(u)$; positive integers B (called the *bin capacity*) and $k<=m$.

Question: Can U be partitioned into k disjoint sets $U_1,...,U_k$ such that for each U_i ($1<=i<=k$) the total sum of the sizes of the items in U_i does not exceed B?

Number: 49

Name: Hitting String [SR12] 3

Input: Finite set $S=\{s^1,...,s^m\}$ each s^i being a *string* of n symbols over $\{0,1,*\}$.

Question: Is there a binary string $x=x_1x_2...x_n$ of length n such that for each s^i in S, s^i and x agree in at least one position.

Number: 50

Name: Rectilinear Picture Compression [SR25] 4-5

Input: An n by n matrix M of *0*s and *1*s; a positive integer K.

Question: Can all of the *1*-valued entries in M be covered by a collection of K or fewer *rectangles*, i.e. is there a sequence of 4-tuples (a_i,b_i,c_i,d_i) (where for all $1<=i<=K$, $a_i<=b_i$ and $c_i<=d_i$) such that:

- For each pair (i,j) ($1<=i,j<=n$) $M[i,j]=1$ if and only if for some k ($1<=k<=K$) it holds that $a_k<=i<=b_k$ and $c_k<=j<=d_k$?

Comments: Despite the apparently complicated definition, this is quite a well-motivated problem. Its background is, as its name suggests, from image compression and the task being set can be interpreted as representing a 2-dimensional image using at most K `blocks' of information.

Number: 51

Name: Sequencing with Release Times and Deadlines [SS1] 4-5

Input: A set T of tasks; for each task t in T: a positive integer *length* $len(t)$; a positive integer *release time* $r(t)$; and a positive integer *deadline* $d(t)$.

Question: Is there a one-processor schedule for the tasks, T, that satisfies the release time constraints *and* meets all of the deadlines, i.e. a one-to-one function *sigma* from the set of tasks to positive integers such that

all of the following hold:

- For any two distinct tasks t and w if $sigma(t)> sigma(w)$ then $sigma(t)>=sigma(w)+len(w)$.
- For all tasks t in T, $sigma(t)>=r(t)$.
- For all tasks t in T, $sigma(t)+len(t)<=d(t)$?

Comments: Another scheduling problem in the style of Problem 8, but in a *single* processor environment. The requirements of the *schedule*, *sigma*, are that it must: 1) assign a unique starting time to each task in the set (*sigma* is a one-to-one function); 2) two tasks cannot be running simultaneously (this is condition (a) above, which says that if a task t is scheduled to start after a task w - $sigma(t)> sigma(w)$ - then the *earliest* time at which t can be scheduled is after w has completed, i.e. $sigma(w)+len(w)$; 3) no task can start before its specified `release time' (condition (b) above); 4) every task has finished no later than its allotted deadline (condition (c) which states that the scheduled starting time of a task t plus the total amount of time that t takes to complete - $len(t)$ - must be no greater than the deadline that is assigned for t, i.e. $d(t)$).

In common with other scheduling and resource management problems there is a large volume of research into heuristics and approximation methods for this problem. Some relevant references may be found in sections of this bibliography.

For a reminder of what a `one-to-one' function is, see Problem 10.

Number: 52

Name: Precedence Constrained Scheduling [SS9] 3-4

Input: A set T of tasks each of which has length *1* or length 2; a partial ordering << on the set of tasks, T; a positive integer deadline D.

Question: Is there a 2-processor schedule, *sigma*, for T that meets the overall deadline D and obeys the `precedence constraints' of the partial order <<, i.e. if $t<< w$ then $sigma(w)>=sigma(t)+len(t)$?

Comments: For a definition of `2-processor schedule' refer to Problem 8. Recall that a *partial order* << on a set R is an ordering relation which satisfies: for all distinct s and t at most one of $s<< t$ or $t<< s$ holds; and for all distinct s, t, w in R if $s<< t$ and $t<< w$ then $s<< w$. The bibliography mentioned above may contain useful references.

Number: 53

Name: Quadratic Congruences [AN1] 3

Input: Positive integers a, b, and c.

Question: Is there a positive integer x whose value is less than c and is such that $x^2==a(mod\ b)$, i.e. the remainder when x^2 is divided by b is equal to a?

Comments: The comments made with respect Problem 9 and Problem 11 are also relevant with respect to this problem.

Number: 54

Name: Square-Tiling [GP13] 5

Input: A set C of n `*colours*'; a collection of *tiles*, T, each tile t being a 4-tuple $< a,b,c,d>$ of colours where a is the colour at the top of the tile; b that on the right-hand side; c that at the bottom; and d that on the left-hand side of the tile; a positive integer $k<=n$.

Question: Is there a *proper tiling* of a k by k square using the tiles in T, i.e. an assignment of a tile $A(i,j)$ in T to each ordered pair (i,j) with $1<=i<=k$, $1<=j<=k$ such that both of the following hold:

- If $A(i,j)=< a,b,c,d>$ and $A(i+1,j)=< e,f,g,h>$ then the colours b and h are identical.
- If $A(i,j)=< a,b,c,d>$ and $A(i,j+1)=< e,f,g,h>$ then the colours c and e are identical?

Comments: Informally, a *proper tiling* is one in which two tiles which are next to each other are required to have the same colour on touching sides, i.e. the right-hand side of one is coloured the same as the left-hand side of its neighbour and similarly with vertically adjacent tiles: the lower side of one is coloured the same as the top side of its neighbour. Decision problems involving questions about tiling patterns tend to be extremely difficult. For the most general form of the question it can be proven that no algorithm at all exists to solve it, i.e. not even an extremely inefficient one. A tiling problem also constitutes the only known `natural' `hard' member of the following class of problems: given a positive integer n as input, determine the number of `objects' of `size' n having a particular property, e.g. the number of n-node graphs, or the number of n-node graphs with a Hamiltonian circuit.

Number: 55

Name: Crossword Puzzle Construction [GP14] 5

Input: A finite set of *characters* $SIGMA=\{sigma_1,...,sigma_k\}$; a finite set, $W=\{w_1,...,w_{2n}\}$ each w_i being a sequence of n characters from *SIGMA*, i.e. a string.

Question: Can an n by n *crossword puzzle* be built using all of the $2n$ words (strings) in W, i.e. if C is an n by n table of `blanks', is there an assignment, f, that maps each entry (i,j) of C to some character in *SIGMA* in such a way that the word formed by taking the n consecutive characters in a row corresponds to a word in W; the word formed by taking the n consecutive characters in a column (reading from top-to-bottom) corresponds to a word in W?

Comments: The definition may look complicated but it isn't. Here is a simple example instance for which a crossword can be constructed. Let $SIGMA=\{A,B,C,D,E,G,O\}$, $n=3$, and $W=\{AGE,AGO,BEG,CAB,CAD,DOG\}$. A 3 by 3 crossword puzzle for this is given by the assignment $C(1,1)=C$; $C(1,2)=A$; $C(1,3)=B$; $C(2,1)=A$; $C(2,2)=G$; $C(2,3)=E$; $C(3,1)=D$; $C(3,2)=O$; $C(3,3)=G$. This results in the grid

```
CAB
AGE
DOG
```

whose correctness is self-evident.

Number: 56

Name: Disjunctive non-tautology [LO8] 2

Input: A A set of *m products* - $P_1,P_2,...,P_m$ - over a set of *n* Boolean valued variables $X_n=<x_1,x_2,...,x_n>$, such that each product depends on exactly three distinct variables from X_n. A **product** being a Boolean expression of the form y_i *AND* y_j *AND* y_k where each *y* is of the form *x* or *-x* (i.e. negation of *x*) with *x* being some variable in X_n.

Question: Is there an assignment of Boolean values to the variables in X_n that results in every product taking the value **false** (equivalently *0*)?

Number: 57

Name: Simultaneous incongruences [AN2] 3

Input: A set of *n* ordered pairs of positive integers $\{(a_1,b_1),...,(a_n,b_n)\}$ where $a_i<=b_i$ for each $1<=i<=n$.

Question: Is there a positive integer *x* such that: for each *i*, a_i *does not* equal the remainder when dividing *x* by b_i?

Comments: As with most number-theoretic problems, the comments regarding Problems 9, 11, and 53 apply.

Number: 58

Name: Betweenness [MS1] 3

Input: A finite set of size *n*, *A*; a set *C* of ordered triples, *(a,b,c)*, of *distinct* elements from *A*.

Question: Is there a one-to-one function, *f:A->{1,2,...,n}* such that for each triple *(a,b,c)* in *C* it holds that either *f(a)< f(b)< f(c)* or *f(c)< f(b)< f(a)*?

Comments: If you need to be reminded of what a one-to-one function is then see Problem 10 to which this problem may appear similar.

Number: 59

Name: Minimum weight and/or graph solution [MS16] 4-5

Input: A *directed acyclic* graph *G(V,A)* having a single node *s*, with no incoming edges; a labelling, *f(v)* of each node having at least one out-going edge in *G*, as either an *and*-node or an *or*-node; a positive integer *K*.

Question: Is there a sub-graph *H(W,B)* of *G(V,A)*, i.e. *W* is a subset of *V* and *B* is a subset of *A* satisfying all of the following:

- *s* is in *W*.
- If *w* is in *W* and *w* is an *and*-node then all of the edges directed out of *w* in *G* belong to *B*.
- If *w* is in *W* and *w* is an *or*-node then at least one of the edges directed out of *w* in *G* belongs to *B*.
- *B* contains at most *K* edges?

Number: 60

Name: Fault-detection in directed graphs. [MS18] 3-4

Input: A *directed, acyclic* graph *G(V,A)* such that *G* has a unique node *t* with no out-going edges; *I* the set of nodes in *V* having no incoming edges; a positive integer *K*.

Question: Is there a `*test set*' of size at most *K* that can detect every `*single fault*' in *G* (**N.B.** not `*every single fault*' but every `*single fault*'), i.e. a subset *T* of the nodes in *I* such that

- *T* contains at most *K* nodes.
- For every node *v* in *V* there exists a node *u* in *T* such that *v* lies on a directed path from *u* to *t* in *G*?

Number: 61

Name: Minimum Broadcast Time. [ND49] 3-4

Input: *n*-node undirected graph *G(V,E)*; a subset V_0 of the nodes in *V*.

Question: Can a message be `broadcast' from the base set V_0 to all of the nodes in *V* in at most *4* steps, i.e. is there a sequence

$V_0;E_1;V_1;E_2;V_2;E_3;V_3;E_4;V_4$

which satisfies all of the following:

- V_i is a subset of *V* for each $0<=i<=4$.
- E_i is a subset of *E* for each $1<=i<=4$.
- $V_4=V$.
- Each edge in E_i has exactly one of its endpoints in V_{i-1} ($1<=i<=4$)
- No two edges in E_i share a common end-point ($1<=i<=4$)
- $V_i=V_{i-1}$*union* $\{w: \{v,w\}$ *in* $E_i\}$ ($1<=i<=4$)?

Number: 62

Name: Disjoint Connecting Paths [ND40] 3-4

Input: *n*-node undirected graph *G(V,E)*; a set of disjoint pairs of nodes $\{(s_1,t_1);...;(s_k,t_k)\}$.

Question: Does *G* contain *k mutually* node disjoint paths, P_i, with P_i being a path from s_i to t_i?

Comments: Mutually disjoint means that no two paths have any common nodes.

Number: 63

Name: Shortest Weight-Constrained Path [ND30] 3

Input: *n*-node undirected graph *G(V,E)*; for each edge *e* in *E* a positive integer length *len(e)* and a positive integer *weight w(e)*; specified nodes *s* and *t* in *V*; positive integers *K* and *W*.

Question: Is there a path from *s* to *t* in *G* that has both:

- Total weight at most *W*.

- Total length at most *K*?

Comments: If all of the edges have the same length or all of the edges have the same weight, then this is simply the normal shortest-path problem for which various efficient algorithms exist, e.g. the Dynamic Programming method of Floyd that is covered in the course. Minimising a single measure, in this context of path lengths, is `easy', but attempting simultaneously to minimise *two* (or more) distinct measures is not.

Number: 64

Name: Minimum Maximal Matching [GT10] 3

Input: *n*-node undirected graph *G(V,E)*; positive integers *k<=|E|*.

Question: Is there a subset, *F*, of at most *k* edges from *E* that forms a maximal matching in *G*, i.e. no two edges in *F* have a common endpoint and every edge of *G* that is not in *F* has a common endpoint with at least one edge of *F*?

Number: 65

Name: Partition into Triangles [GT11] 3

Input: *(3n)*-node undirected graph *G(V,E)*.

Question: Can the nodes of *G* be partitioned into *n* disjoint sets - $V_1,...,V_n$ - each of which contains exactly *3* nodes and is such that for each $V_i=\{u_i,v_i,w_i\}$, all three of the edges $\{u_i,v_i\}$, $\{u_i,w_i\}$ and $\{v_i,w_i\}$ belong to *E*?

Number: 66

Name: Partition into Forests [GT14] 3

Input: *n*-node undirected graph *G(V,E)*; positive integer *k<=n*

Question: Can the nodes of *G* be partitioned into $t<=k$ disjoint sets $V_1,...,V_t$ - in such a way that for each V_i ($1<=i<=t$), the subgraph $G_i(V_i,E_i)$ induced by V_i contains no cycles, i.e. is a forest (set of trees)?

Comments: If *G(V,E)* is a graph, then the *subgraph induced* by a subset *W* of *V* is the graph *H(W,F)* whose edges, *F*, are formed by all the edges in *E* that connect two nodes in *W*.

Number: 67

Name: Partition into Cliques [GT15] 3

Input: *n*-node undirected graph *G(V,E)*; positive integer *k<=n*

Question: Can the nodes of *G* be partitioned into $t<=k$ disjoint sets - $V_1,...,V_t$ - in such a way that for each V_i ($1<=i<=t$), the subgraph $G_i(V_i,E_i)$ induced by V_i is a clique, i.e. a graph in which every pair of nodes is connected by an edge?

Comments: For definition of *induced subgraph* see Problem 66.

Number: 68

Name: Partition into Perfect Matchings [GT16] 3

Input: *n*-node undirected graph *G(V,E)*; positive integer *k<=n*

Question: Can the nodes of *G* be partitioned into $t<=k$ disjoint sets - $V_1,...,V_t$ - in such a way that for each V_i ($1<=i<=t$), the subgraph $G_i(V_i,E_i)$ induced by V_i is a *perfect matching*, i.e. a graph in which every node is the endpoint of exactly one edge?

Comments: For definition of *induced subgraph* see Problem 66.

Number: 69

Name: Covering by cliques [GT17] 3

Input: *n*-node undirected graph *G(V,E)*; positive integer *k<=n*

Question: Are there $t<=k$ subsets - $V_1,...,V_t$ of *V* - such that both of the following hold:

- The subgraph $G_i(V_i,E_i)$ of *G(V,E)* induced by V_i is a clique.
- For each edge $\{u,v\}$ in *E* there is some subset V_i that contains *both u* and *v*?

Comments: For relevant definitions see Problems 66 and 67.

Number: 70

Name: Degree-bounded connected subgraph [GT26] 3

Input: *n*-node undirected graph *G(V,E)*; non-negative integer *d<=n*; positive integer *k<=|E|*.

Question: Is there a subset *F* of the edges in *E* having size at least *k* and such that subgraph *G(V,F)* of *G* is connected and has no node of degree greater than *d*?

Comments: For the definition of `degree of a node' see Problem 7. A graph is *connected* if for every pair of nodes *v* and *w* there is a path connecting *v* and *w* in the graph.

Number: 71

Name: Uniconnected subgraph [GT30] 3

Input: *n*-node *directed* graph *G(V,A)*; positive integer *k<=|A|*.

Question: Is there a subset *B* of the edges in *A* having size at least *k* and such that the subgraph *H(V,B)* of *G* has *at most* one directed path between any pair of nodes in *V*?

Number: 72

Name: Minimum *k*-connected subgraph [GT31] 3

Input: *n*-node undirected graph *G(V,E)*; positive integers *k<=n* and *B<=|E|*.

Question: Is there a subset *F* of the edges *E* having size at most *B* and such that the subgraph *H(V,F)* of *G* is *k-connected*, i.e. remains connected after removing any set *W* of a most *k-1* nodes and their adjacent edges?

Comments: For a reminder of what a `connected graph' is see Problem 70.

Number: 73

Name: Minimum cut linear arrangement [GT44] 3

Input: *n*-node undirected graph *G(V,E)*; positive integer *k*.

Question: Is there a one-to-one function *f:V->{1,2,...,n}* such that for all *i*, with *1< i< n*, the total number of edges *{u,v}* in *E* for which *f(u)<=i< f(v)* is at most *k*?

Comments: To rediscover what a `one-to-one' function is, see Problem 10.

Number: 74

Name: Consecutive Sets [SR18] 4

Input: Finite alphabet SIGMA of t symbols; collection C={$SIGMA_1$,$SIGMA_2$,...,$SIGMA_n$} of subsets of SIGMA; positive integer K.

Question: Is there a string, w, (sequence of symbols) over SIGMA such that |w|< =K and for each 1< =i< =n, all of the symbols in $SIGMA_i$ appear in a consecutive block of |$SIGMA_i$| symbols in the string w?

Comments: N.B. minor typo. in Garey and Johnson, W written instead of w.

Number: 75

Name: Multiple choice matching [GT55] 3

Input: *n*-node undirected graph *G(V,E)*; partition of *E* into *t* disjoint sets - E_1,...,E_t; positive integer *k*.

Question: Is there a subset *F* of *E* having size at least *k* and satisfying both of the following:

- No two edges in *F* share a common endpoint.
- *F* contains at most one edge from each E_i (*1<=i<=t*)?

Number: 76

Name: Path distinguishers [GT60] 3

Input: *n*-node *directed, acyclic* graph *G(V,A)*; specified nodes *s* and *t* in *V*; positive integer *k<=|A|*.

Question: Is there a subset *B* of the edges *A* having size at most *k* and such that for any pair of distinct paths *P* and *Q* from *s* to *t* in *G*, there is an edge in *B* that is in *exactly* one of *P* and *Q*?

Number: 77

Name: Nestril-Rodl dimension [GT62] 5

Input: *n*-node undirected graph *G(V,E)*; positive integer *k<=n*.

Question: Is there a one-to-one function

f:V-> {(a_1, a_2,...,a_k) : 1 <= a_i<= n }

such that for all pairs of nodes *u* and *v* in *V*, *{u,v}* is an edge of *G* if and only if *f(u)* and *f(v) disagree* in all *k* components?

Number: 78

Name: Shortest total path length spanning tree [ND3] 4-5

Input: *n*-node undirected graph *G(V,E)*; positive integer *B*.

Question: Is there a spanning tree *T(V,F)* of *G* such that the sum over all pairs of nodes *u* and *v* of the length of the path between *u* and *v* in *T* is no greater than *B*?

Comments: For the definition of spanning tree, see Problem 21.

Number: 79

Name: Induced Path [GT23] 3

Input: *n*-node undirected graph *G(V,E)*; positive integer *k<=n*.

Question: Is there a subset *W* of the nodes *V* having size at least *k* and such that the subgraph *H(W,F)* of *G* induced by *W* is a simple path on *|W|* nodes?

Comments: For relevant definitions refer to Problem 66.

Number: 80

Name: Bounded Component Spanning Forest [ND10] 3

Input: *n*-node undirected graph *G(V,E)*; positive integer *k<=n*.

Question: Can *V* be partitioned into *t* disjoint sets - V_1,...,V_t - where *t<=k*, such that for each *i* (*1<=i<=t*) both of the following hold:

- The subgraph *H(V_i,E_i)* of *G* induced by V_i is connected.
- The number of nodes in V_i is at most 4.

Comments: For relevant definitions refer to Problem 66 and Problem 70.

Number: 81

Name: Maximum subgraph matching [GT50] 3

Input: *Directed* graphs *G(V,A)*, *H(W,B)*; positive integer *k*.

Question: Is there a subset *R* of *VxW*, i.e. pairs of nodes where one node is from *V* and one from *W*, having size at least *k* and such that for every < *u,v*> and < *x,y*> in *R* the edge *(u,x)* belongs to *A* if and only if the edge *(v,y)* belongs to *B*?

Number: 82

Name: Numerical 3-dimensional matching [SP16] 3

Input: Disjoint sets *W*, *X*, and *Y* each containing exactly *m* elements; for each element *u* of *W union X union Y*, a positive integer size *s(u)*; a positive integer bound *B*.

Question: Can the *3m* element set *W union X union Y* be partitioned into *m* disjoint sets A_1,...,A_m in such a way that:

- Each A_i contains exactly one element from *W*, exactly one element from *X*, and exactly one element from *Y*.
- Each A_i has total size exactly equal to *B*?

Number: 83

Name: Open Hemisphere [MP6] 4-5

Input: Finite set *X* of *m*-tuples of integers; a positive integer *k<=|X|*.

Question: Is there an *m*-tuple *y* of *rational* numbers such that *sum from i=1 to m* x_iy_i<> *0* for at least *k* of the *m*-tuples in *X*, x_i (resp. y_i) being the *i*th component of the *m*-tuple *x* (resp. *y*)?

Number: 84

Name: Largest Common Subgraph [GT49] 3-4

Input: 2 *n*-node undirected graphs *G(V,E)*, *H(W,F)*; positive integer *k*.

Question: Do there exist subsets E_1 of the edges *E* and F_1 of the edges *F* that satisfy all of the following:

- *|E_1|=|F_1|*.
- *|E_1|>=k*, *|F_1|>=k*.
- The graphs *G(V,E_1)* and *H(W,F_1)* are *isomorphic*, i.e. there is a mapping *f* between *V* and *W* such that *{x,y}* is an edge in E_1 if and only if *{f(x),f(y)}* is an edge in F_1?

Number: 85

Name: Clustering [MS9] 3

Input: Finite set *X*; for each pair of elements *x* and *y* in *X*, a positive integer distance *d(x,y)*; positive integer

B.

Question: Is there a partition of *X* into *3* disjoint sets - X_1,X_2,X_3 - with which: for each set X_i (*1<=i<=3*), for all pairs *x* and *y* in X_i it holds that *d(x,y)<=B*?

Number: 86

Name: Protein Folding 5

Input: Three positive integers, *t*, *n*, and *k*; a finite alphabet, *SIGMA*, of *t* symbols; a string *S* of length *n*.

Question: Is there an embedding of *S* into a two dimensional grid that has a bond-score of at least *k*? i.e. if *S=s_1s_2...s_n*, is there a one-to-one function *FOLD:{1,...,n}->***N**x**N** such that:

- For all *1 <= i< n* if *FOLD(i)=(x,y)* and *FOLD (i+1)=(w,z)*, then either *{x=w and |y-z|=1}* or *{|x-w|=1 and y=z}*. (i.e. consecutive members of *S* are mapped onto adjacent grid co-ordinates).
- The number of *pairs {(x,y),(w,z)}* of grid positions for which *both*
- *(x,y)* and *(w,z)* are neighbours, (i.e. *{x=w and |y-z|=1}* or *{|x-w|=1 and y=z}*).
- *FOLD(i)=(x,y)* and *FOLD(j)=(w,z)* (where *i* and *j* satisfy *1<=i< j<=n*), and s_i=s_j, i.e. the symbols (in *SIGMA*) mapped onto *(x,y)* and *(w,z)* are identical.
- is at least *k*?

Comments: The problem arises from an extremely simplified model of protein-folding of interest in computational biology. The *NP*-completeness proof is highly non-trivial (by a transformation from *3-SAT*), is a recent result not mentioned in Garey and Johnson, and is due to Paterson and Przytycka (1996). It is important to note that the alphabet is part of the input. It is an open question as to whether the variant in which an alphabet of a fixed size, e.g. 2 as with binary, remains *NP*-complete. In should also be remembered that no limit is placed on the grid *size*, although trivially, an *n x n* grid will always suffice.

Number: 87

Name: Hamming Centre 3-4

Input: A set *S* of *k* binary strings, each of length *n*; a positive integer *r*.

Question: Is there an *n*-bit string *y* such that for every string *x* in *S*, the *Hamming distance*, *H(x,y)* is at most *r*?.

Comments: The Hamming distance, *H(x,y)*, between two binary strings *x* and *y* of length *n*, is the number of bit positions in which *x* and *y* have differing values.

This is another recent result not given in Garey and Johnson, its *NP*-complete classification being given in Frances and Litman (1997).

M. Frances and A. Litman. On Covering Problems of Codes, *Theory of Computing Systems*, 30, 1997, 113-119

Number: 88

Name: 2-Spanner 3

Input: An undirected graph *G(V,E)*; a positive integer *s*.

Question: Does *G(V,E)* contain a *2-spanner* with at most *s* edges, i.e. subset *F* of the edges *E* having size at most *s* and such that for each edge *{v,w}* in *E* either *{v,w}* is in *F* or (for some node *x* in *V*) the edges *{v,x}* and *{x,w}* are both in *F*?

Comments: 2-Spanners (or more generally *t*-spanners for some positive integer *t*) are a natural extension of the idea of spanning tree (**N.B.** A spanner is required neither to be a tree nor, even, a connected graph), and were first considered in the context of problems in communications networks and parallel architectures by Peleg and Ullman (1989). Subsequently, Peleg and Schaffer (1989) showed this problem to be *NP*-complete, hence its absence from the appendix in Garey and Johnson.

D. Peleg and A.A. Schaffer. Graph Spanners. *Jnl. of Graph Theory*, 13, (1989), 99-116.

D. Peleg and J.D. Ullman. An Optimal Synchroniser for the Hypercube. *SIAM Jnl. on Computing*, 18, (1989), 740-747

Chapter 13

Some $\mathcal{NP}$-Complete Problems

13.1 Statements of the Problems

In this chapter we will show that certain classical algorithmic problems are $\mathcal{NP}$-complete.

This chapter is heavily inspired by Lewis and Papadimitriou's excellent treatment [**?**].

In order to study the complexity of these problems in terms of resource (time or space) bounded Turing machines (or RAM programs), it is crucial to be able to encode instances of a problem P as strings in a language L_P.

Then an instance of a problem P is solvable iff the corresponding string belongs to the language L_P.

This implies that our problems must have a yes–no answer, which is not always the usual formulation of optimization problems where what is required is to find some *optimal* solution, that is, a solution minimizing or maximizing so objective (cost) function F.

For example the standard formulation of the traveling salesman problem asks for a tour (of the cities) of minimal cost.

Fortunately, there is a trick to reformulate an optimization problem as a yes–no answer problem, which is to explicitly incorporate a *budget* (or *cost*) term B into the problem, and instead of asking whether some objective function F has a minimum or a maximum w, we ask whether there is a solution w such that $F(w) \leq B$ in the case of a minimum solution, or $F(w) \geq B$ in the case of a maximum solution.

We will see several examples of this technique in Problems 5–8 listed below.

The problems that will consider are

(1) Exact Cover

(2) Hamiltonian Cycle for directed graphs

(3) Hamiltonian Cycle for undirected graphs

(4) The Traveling Salesman Problem

(5) Independent Set

(6) Clique

(7) Node Cover

(8) Knapsack, also called subset sum

(9) Inequivalence of $*$-free Regular Expressions

(10) The 0-1-integer programming problem

We begin by describing each of these problems.

(1) **Exact Cover**

We are given a finite nonempty set $U = \{u_1, \ldots, u_n\}$ (the *universe*), and a family $\mathcal{F} = \{S_1, \ldots, S_m\}$ of $m \geq 1$ *nonempty subsets* of U.

The question is whether there is an *exact cover*, that is, a subfamily $\mathcal{C} \subseteq \mathcal{F}$ of subsets in $\mathcal{F}$ such that the sets in $\mathcal{C}$ are disjoint and their union is equal to U.

For example, let
$U = \{u_1, u_2, u_3, u_4, u_5, u_6\}$, and let $\mathcal{F}$ be the family

$$\mathcal{F} = \{\{u_1, u_3\}, \{u_2, u_3, u_6\}, \{u_1, u_5\}, \{u_2, u_3, u_4\}, \{u_5, u_6\}, \{u_2, u_4\}\}.$$

The subfamily

$$\mathcal{C} = \{\{u_1, u_3\}, \{u_5, u_6\}, \{u_2, u_4\}\}$$

is an exact cover.

It is easy to see that **Exact Cover** is in $\mathcal{NP}$.

To prove that it is $\mathcal{NP}$-complete, we will reduce the **Satisfiability Problem** to it.

This means that we provide a method running in polynomial time that converts every instance of the **Satisfiability Problem** to an instance of **Exact Cover**, such that the first problem has a solution iff the converted problem has a solution.

(2) **Hamiltonian Cycle (for Directed Graphs)**

Recall that a *directed graph* G is a pair $G = (V, E)$, where $E \subseteq V \times V$.

Elements of V are called *nodes* (or *vertices*). A pair $(u, v) \in E$ is called an *edge* of G.

We will restrict ourselves to *simple graphs*, that is, graphs without edges of the form (u, u);

equivalently, $G = (V, E)$ is a simple graph if whenever $(u, v) \in E$, then $u \neq v$.

Given any two nodes $u, v \in V$, a *path from u to v* is any sequence of $n + 1$ edges ($n \geq 0$)

$$(u, v_1), (v_1, v_2), \ldots, (v_n, v).$$

(If $n = 0$, a path from u to v is simply a single edge, (u, v).)

A directed graph G is *strongly connected* if for every pair $(u, v) \in V \times V$, there is a path from u to v. A *closed path, or cycle*, is a path from some node u to itself.

We will restrict out attention to finite graphs, i.e. graphs (V, E) where V is a finite set.

Definition 13.1. Given a directed graph G, a *Hamiltonian cycle* is a cycle that passes through all the nodes exactly once (note, some edges may not be traversed at all).

Hamiltonian Cycle Problem (for Directed Graphs): Given a directed graph G, is there an Hamiltonian cycle in G?

Is there is a Hamiltonian cycle in the directed graph D shown in Figure 13.1?

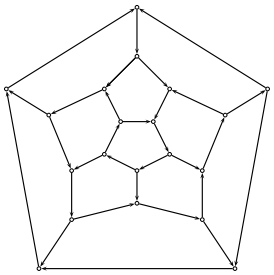


Figure 13.1: A tour "around the world."

Finding a Hamiltonian cycle in this graph does not appear to be so easy! A solution is shown in Figure 13.2 below.

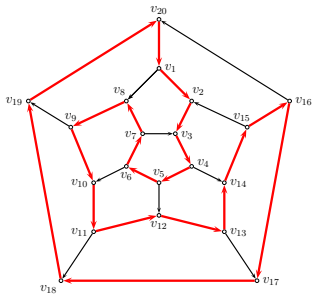


Figure 13.2: A Hamiltonian cycle in D.

It is easy to see that **Hamiltonian Cycle (for Directed Graphs)** is in $\mathcal{NP}$.

To prove that it is $\mathcal{NP}$-complete, we will reduce **Exact Cover** to it.

This means that we provide a method running in polynomial time that converts every instance of **Exact Cover** to an instance of **Hamiltonian Cycle (for Directed Graphs)** such that the first problem has a solution iff the converted problem has a solution. This is perphaps the hardest reduction.

(3) **Hamiltonian Cycle (for Undirected Graphs)**

Recall that an *undirected graph* G is a pair $G = (V, E)$, where E is a set of subsets $\{u, v\}$ of V consisting of exactly two distinct elements.

Elements of V are called *nodes* (or *vertices*). A pair $\{u, v\} \in E$ is called an *edge* of G.

Given any two nodes $u, v \in V$, a *path from u to v* is any sequence of n nodes ($n \geq 2$)

$$u = u_1, u_2, \ldots, u_n = v$$

such that $\{u_i, u_{i+1}\} \in E$ for $i = 1, \ldots, n-1$. (If $n = 2$, a path from u to v is simply a single edge, $\{u, v\}$.)

An undirected graph G is *connected* if for every pair $(u, v) \in V \times V$, there is a path from u to v.

A *closed path, or cycle*, is a path from some node u to itself.

Definition 13.2. Given an undirected graph G, a *Hamiltonian cycle* is a cycle that passes through all the nodes exactly once (note, some edges may not be traversed at all).

Hamiltonian Cycle Problem (for Undirected Graphs): Given an undirected graph G, is there an Hamiltonian cycle in G?

An instance of this problem is obtained by changing every directed edge in the directed graph of Figure 13.1 to an undirected edge.

The directed Hamiltonian cycle given in Figure 13.2 is also an undirected Hamiltonian cycle of the undirected graph of Figure 13.3.

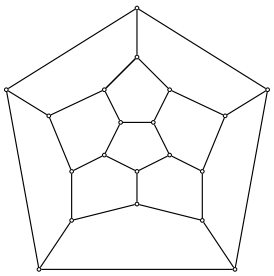


Figure 13.3: A tour "around the world," undirected version.

We see immediately that **Hamiltonian Cycle (for Undirected Graphs)** is in $\mathcal{NP}$.

To prove that it is $\mathcal{NP}$-complete, we will reduce **Hamiltonian Cycle (for Directed Graphs)** to it.

This means that we provide a method running in polynomial time that converts every instance of **Hamiltonian Cycle (for Directed Graphs)** to an instance of **Hamiltonian Cycle (for Undirected Graphs)** such that the first problem has a solution iff the converted problem has a solution. This is an easy reduction.

(4) **Traveling Salesman Problem**

We are given a set $\{c_1, c_2, \ldots, c_n\}$ of $n \geq 2$ cities, and an $n \times n$ matrix $D = (d_{ij})$ of nonnegative integers, where d_{ij} is the *distance* (or *cost*) of traveling from city c_i to city c_j.

We assume that $d_{ii} = 0$ and $d_{ij} = d_{ji}$ for all i, j, so that the matrix D is symmetric and has zero diagonal.

Traveling Salesman Problem: Given some $n \times n$ matrix $D = (d_{ij})$ as above and some integer $B \geq 0$ (the *budget* of the traveling salesman), find a permutation π of $\{1, 2, \ldots, n\}$ such that

$$c(\pi) = d_{\pi(1)\pi(2)} + d_{\pi(2)\pi(3)} + \cdots + d_{\pi(n-1)\pi(n)} + d_{\pi(n)\pi(1)} \leq B.$$

The quantity $c(\pi)$ is the *cost* of the trip specified by π.

The Traveling Salesman Problem has been stated in terms of a budget so that it has a yes or no answer, which allows us to convert it into a language. A minimal solution corresponds to the smallest feasible value of B.

Example 13.1. Consider the 4×4 symmetric matrix given by

$$D = \begin{pmatrix} 0 & 2 & 1 & 1 \\ 2 & 0 & 1 & 1 \\ 1 & 1 & 0 & 3 \\ 1 & 1 & 3 & 0 \end{pmatrix},$$

and the budget $B = 4$. The tour specified by the permutation

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 4 & 2 & 3 \end{pmatrix}$$

has cost 4, since

$$\begin{aligned} c(\pi) &= d_{\pi(1)\pi(2)} + d_{\pi(2)\pi(3)} + d_{\pi(3)\pi(4)} + d_{\pi(4)\pi(1)} \\ &= d_{14} + d_{42} + d_{23} + d_{31} \\ &= 1 + 1 + 1 + 1 = 4. \end{aligned}$$

The cities in this tour are traversed in the order

$$(1, 4, 2, 3, 1).$$

It is clear that the **Traveling Salesman Problem** is in $\mathcal{NP}$.

To show that it is $\mathcal{NP}$-complete, we reduce the **Hamiltonian Cycle Problem (Undirected Graphs)** to it.

This means that we provide a method running in polynomial time that converts every instance of **Hamiltonian Cycle Problem (Undirected Graphs)** to an instance of the **Traveling Salesman Problem** such that the first problem has a solution iff the converted problem has a solution.

(5) **Independent Set**

The problem is this: Given an undirected graph $G = (V, E)$ and an integer $K \geq 2$, is there a set C of nodes with $|C| \geq K$ such that for all $v_i, v_j \in C$, there is *no* edge $\{v_i, v_j\} \in E$?

A maximal independent set with 3 nodes is shown in Figure 13.4.

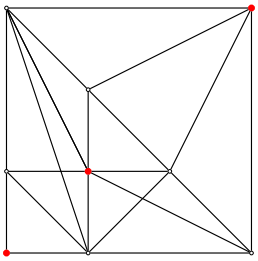


Figure 13.4: A maximal Independent Set in a graph

A maximal solution corresponds to the largest feasible value of K.

The problem **Independent Set** is obviously in $\mathcal{NP}$.

To show that it is $\mathcal{NP}$-complete, we reduce **Exact** 3**-Satisfiability** to it.

This means that we provide a method running in polynomial time that converts every instance of **Exact** 3**-Satisfiability** to an instance of **Independent Set** such that the first problem has a solution iff the converted problem has a solution.

(6) **Clique**

The problem is this: Given an undirected graph $G = (V, E)$ and an integer $K \geq 2$, is there a set C of nodes with $|C| \geq K$ such that for all $v_i, v_j \in C$, there is *some* edge $\{v_i, v_j\} \in E$?

Equivalently, does G contain a complete subgraph with at least K nodes?

A maximal clique with 4 nodes is shown in Figure 13.5.

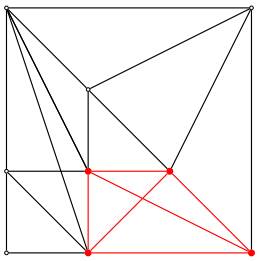


Figure 13.5: A maximal Clique in a graph

A maximal solution corresponds to the largest feasible value of K.

The problem **Clique** is obviously in $\mathcal{NP}$.

To show that it is $\mathcal{NP}$-complete, we reduce **Independent Set** to it.

This means that we provide a method running in polynomial time that converts every instance of **Independent Set** to an instance of **Clique** such that the first problem has a solution iff the converted problem has a solution.

(7) **Node Cover**

The problem is this: Given an undirected graph $G = (V, E)$ and an integer $B \geq 2$, is there a set C of nodes with $|C| \leq B$ such that C *covers all edges in* G, which means that for every edge $\{v_i, v_j\} \in E$, either $v_i \in C$ or $v_j \in C$?

A minimal node cover with 6 nodes is shown in Figure 13.6.

A minimal solution corresponds to the smallest feasible value of B.

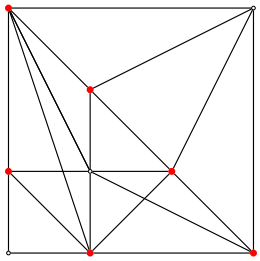


Figure 13.6: A minimal Node Cover in a graph

The problem **Node Cover** is obviously in $\mathcal{NP}$.

To show that it is $\mathcal{NP}$-complete, we reduce **Independent Set** to it.

This means that we provide a method running in polynomial time that converts every instance of **Independent Set** to an instance of **Node Cover** such that the first problem has a solution iff the converted problem has a solution.

The Node Cover problem has the following interesting interpretation:

think of the nodes of the graph as rooms of a museum (or art gallery *etc.*), and each edge as a straight corridor that joins two rooms.

Then Node Cover may be useful in assigning as few as possible guards to the rooms, so that all corridors can be seen by a guard.

(8) **Knapsack (also called Subset sum)**

The problem is this: Given a finite nonempty set $S = \{a_1, a_2, \ldots, a_n\}$ of nonnegative integers, and some integer $K \geq 0$, all represented in binary, is there a nonempty subset $I \subseteq \{1, 2, \ldots, n\}$ such that

$$\sum_{i \in I} a_i = K?$$

A "concrete" realization of this problem is that of a hiker who is trying to fill her/his backpack to its maximum capacity with items of varying weights or values.

It is easy to see that the **Knapsack** Problem is in $\mathcal{NP}$.

To show that it is $\mathcal{NP}$-complete, we reduce **Exact Cover** to it.

This means that we provide a method running in polynomial time that converts every instance of **Exact Cover** to an instance of **Knapsack** Problem such that the first problem has a solution iff the converted problem has a solution.

Remark: The **0-1 Knapsack Problem** is defined as the following problem.

Given a set of n items, numbered from 1 to n, each with a *weight* $w_i \in \mathbb{N}$ and a *value* $v_i \in \mathbb{N}$, given a maximum capacity $W \in \mathbb{N}$ and a budget $B \in \mathbb{N}$, is there a set of n variables $x_1, \ldots, x_n$ with $x_i \in \{0, 1\}$ such that

$$\sum_{i=1}^{n} x_i v_i \geq B,$$
$$\sum_{i=1}^{n} x_i w_i \leq W.$$

Informally, the problem is to pick items to include in the knapsack so that the sum of the values exceeds a given minimum B (the goal is to maximize this sum), and the sum of the weights is less than or equal to the capacity W of the knapsack.

A maximal solution corresponds to the largest feasible value of B.

The **Knapsack** Problem as we defined it (which is how Lewis and Papadimitriou define it) is the special case where $v_i = w_i = 1$ for $i = 1, \ldots, n$ and $W = B$.

For this reason, it is also called the **Subset Sum** Problem.

Clearly, the **Knapsack** (**Subset Sum**) Problem reduces to the **0-1 Knapsack** Problem, and thus the **0-1 Knapsack** Problem is also NP-complete.

(9) **Inequivalence of $*$-free Regular Expressions**

Recall that the problem of deciding the equivalence $R_1 \cong R_2$ of two regular expressions R_1 and R_2 is the problem of deciding whether R_1 and R_2 define the same language, that is, $\mathcal{L}[R_1] = \mathcal{L}[R_2]$.

Is this problem in $\mathcal{NP}$?

In order to show that the equivalence problem for regular expressions is in $\mathcal{NP}$ we would have to be able to somehow check in polynomial time that two expressions define the same language, but this is still an open problem.

What might be easier is to decide whether two regular expressions R_1 and R_2 are *inequivalent*.

For this, we just have to find a string w such that either $w \in \mathcal{L}[R_1] - \mathcal{L}[R_2]$ or $w \in \mathcal{L}[R_2] - \mathcal{L}[R_1]$.

The problem is that if we can guess such a string w, we still have to check in polynomial time that $w \in (\mathcal{L}[R_1] - \mathcal{L}[R_2]) \cup (\mathcal{L}[R_2] - \mathcal{L}[R_1])$, and this implies that there is a bound on the length of w which is polynomial in the sizes of R_1 and R_2.

Again, this is an open problem.

To obtain a problem in $\mathcal{NP}$ we have to consider a restricted type of regular expressions, and it turns out that $*$-free regular expressions are the right candidate.

A *$*$-free regular expression* is a regular expression which is built up from the atomic expressions using only $+$ and $\cdot$, but not $*$. For example,

$$R = ((a + b)aa(a + b) + aba(a + b)b)$$

is such an expression.

It is easy to see that if R is a $*$-free regular expression, then for every string $w \in \mathcal{L}[R]$ we have $|w| \leq |R|$. In particular, $\mathcal{L}[R]$ is finite.

The above observation shows that if R_1 and R_2 are $*$-free and if there is a string $w \in (\mathcal{L}[R_1] - \mathcal{L}[R_2]) \cup (\mathcal{L}[R_2] - \mathcal{L}[R_1])$, then $|w| \leq |R_1| + |R_2|$, so we can indeed check this in polynomial time.

It follows that the inequivalence problem for $*$ -free regular expressions is in $\mathcal{NP}$.

To show that it is $\mathcal{NP}$-complete, we reduce the **Satisfiability Problem** to it.

This means that we provide a method running in polynomial time that converts every instance of **Satisfiability Problem** to an instance of **Inequivalence of Regular Expressions** such that the first problem has a solution iff the converted problem has a solution.

(10) 0-1 **integer programming problem**

Let A be any $p \times q$ matrix with integer coefficients and let $b \in \mathbb{Z}^p$ be any vector with integer coefficients.

The 0-1 **integer programming problem** is to find whether a system of p linear equations in q variables

$$\begin{aligned} a_{11}x_1 + \cdots + a_{1q}x_q &= b_1 \\ &\vdots \qquad\qquad \vdots \\ a_{i1}x_1 + \cdots + a_{iq}x_q &= b_i \\ &\vdots \qquad\qquad \vdots \\ a_{p1}x_1 + \cdots + a_{pq}x_q &= b_p \end{aligned}$$

with $a_{ij}, b_i \in \mathbb{Z}$ has any solution $x \in \{0,1\}^q$, that is, with $x_i \in \{0,1\}$.

In matrix form, if we let

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1q} \\ \vdots & \ddots & \vdots \\ a_{p1} & \cdots & a_{pq} \end{pmatrix}, \quad b = \begin{pmatrix} b_1 \\ \vdots \\ b_p \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ \vdots \\ x_q \end{pmatrix},$$

then we write the above system as

$$Ax = b.$$

It is immediate that 0-1 **integer programming problem** is in $\mathcal{NP}$.

To prove that it is $\mathcal{NP}$-complete we reduce the **bounded tiling** problem to it.

This means that we provide a method running in polynomial time that converts every instance of the **bounded tiling** problem to an instance of the 0-1 **integer programming problem** such that the first problem has a solution iff the converted problem has a solution.

13.2 Proofs of $\mathcal{NP}$-Completeness

(1) **Exact Cover**

To prove that **Exact Cover** is $\mathcal{NP}$-complete, we reduce the **Satisfiability Problem** to it:

Satisfiability Problem $\leq_P$ **Exact Cover**

Given a set $F = \{C_1, \ldots, C_\ell\}$ of ℓ clauses constructed from n propositional variables $x_1, \ldots, x_n$, we must construct in polynomial time an instance $\tau(F) = (U, \mathcal{F})$ of **Exact Cover** such that F is satisfiable iff $\tau(F)$ has a solution.

Example 13.2. If

$$F = \{C_1 = (x_1 \vee \overline{x_2}),\ C_2 = (\overline{x_1} \vee x_2 \vee x_3),\ C_3 = (x_2),\\ C_4 = (\overline{x_2} \vee \overline{x_3})\},$$

then the universe U is given by

$$U = \{x_1, x_2, x_3, C_1, C_2, C_3, C_4, p_{11}, p_{12}, p_{21}, p_{22}, p_{23}, p_{31},\\ p_{41}, p_{42}\},$$

and the family $\mathcal{F}$ consists of the subsets

$\{p_{11}\}, \{p_{12}\}, \{p_{21}\}, \{p_{22}\}, \{p_{23}\}, \{p_{31}\}, \{p_{41}\}, \{p_{42}\}$
$T_{1,\mathbf{F}} = \{x_1, p_{11}\}$
$T_{1,\mathbf{T}} = \{x_1, p_{21}\}$
$T_{2,\mathbf{F}} = \{x_2, p_{22}, p_{31}\}$
$T_{2,\mathbf{T}} = \{x_2, p_{12}, p_{41}\}$
$T_{3,\mathbf{F}} = \{x_3, p_{23}\}$
$T_{3,\mathbf{T}} = \{x_3, p_{42}\}$
$\{C_1, p_{11}\}, \{C_1, p_{12}\}, \{C_2, p_{21}\}, \{C_2, p_{22}\}, \{C_2, p_{23}\},$
$\{C_3, p_{31}\}, \{C_4, p_{41}\}, \{C_4, p_{42}\}.$

It is easy to check that the set $\mathcal{C}$ consisting of the following subsets is an exact cover:

$T_{1,\mathbf{T}} = \{x_1, p_{21}\}, T_{2,\mathbf{T}} = \{x_2, p_{12}, p_{41}\}, T_{3,\mathbf{F}} = \{x_3, p_{23}\},$
$\{C_1, p_{11}\}, \{C_2, p_{22}\}, \{C_3, p_{31}\}, \{C_4, p_{42}\}.$

The general method to construct $(U, \mathcal{F})$ from $F = \{C_1, \ldots, C_\ell\}$ proceeds as follows. Say

$$C_j = (L_{j1} \vee \cdots \vee L_{jm_j})$$

is the jth clause in F, where L_{jk} denotes the kth literal in C_j and $m_j \geq 1$. The universe of $\tau(F)$ is the set

$$U = \{x_i \mid 1 \leq i \leq n\} \cup \{C_j \mid 1 \leq j \leq \ell\}\\ \cup \{p_{jk} \mid 1 \leq j \leq \ell,\ 1 \leq k \leq m_j\}$$

where in the third set p_{jk} corresponds to the kth literal in C_j.

The following subsets are included in $\mathcal{F}$:

(a) There is a set $\{p_{jk}\}$ for every p_{jk}.

(b) For every boolean variable x_i, the following two sets are in $\mathcal{F}$:

$$T_{i,\mathbf{T}} = \{x_i\} \cup \{p_{jk} \mid L_{jk} = \overline{x_i}\}$$

which contains x_i and all negative occurrences of x_i, and

$$T_{i,\mathbf{F}} = \{x_i\} \cup \{p_{jk} \mid L_{jk} = x_i\}$$

which contains x_i and all its positive occurrences. Note carefully that $T_{i,\mathbf{T}}$ involves negative occurrences of x_i whereas $T_{i,\mathbf{F}}$ involves positive occurrences of x_i.

(c) For every clause C_j, the m_j sets $\{C_j, p_{jk}\}$ are in $\mathcal{F}$.

It remains to prove that F is satisfiable iff $\tau(F)$ has a solution.

We claim that if v is a truth assignement that satisfies F, then we can make an exact cover $\mathcal{C}$ as follows:

For each x_i, we put the subset $T_{i,\mathbf{T}}$ in $\mathcal{C}$ iff $v(x_i) = \mathbf{T}$, else we we put the subset $T_{i,\mathbf{F}}$ in $\mathcal{C}$ iff $v(x_i) = \mathbf{F}$.

Also, for every clause C_j, we put some subset $\{C_j, p_{jk}\}$ in $\mathcal{C}$ for a literal L_{jk} which is made true by v.

By construction of $T_{i,\mathbf{T}}$ and $T_{i,\mathbf{F}}$, this p_{jk} is not in any set in $\mathcal{C}$ selected so far. Since by hypothesis F is satisfiable, such a literal exists for every clause.

Having covered all x_i and C_j, we put a set $\{p_{jk}\}$ in $\mathcal{C}$ for every remaining p_{jk} which has not yet been covered by the sets already in $\mathcal{C}$.

Going back to Example 13.2, the truth assigment $v(x_1) = \mathbf{T}, v(x_2) = \mathbf{T}, v(x_3) = \mathbf{F}$ satisfies

$$F = \{C_1 = (x_1 \vee \overline{x_2}),\ C_2 = (\overline{x_1} \vee x_2 \vee x_3),\ C_3 = (x_2),\\ C_4 = (\overline{x_2} \vee \overline{x_3})\},$$

so we put

$$T_{1,\mathbf{T}} = \{x_1, p_{21}\}, T_{2,\mathbf{T}} = \{x_2, p_{12}, p_{41}\}, T_{3,\mathbf{F}} = \{x_3, p_{23}\},\\ \{C_1, p_{11}\}, \{C_2, p_{22}\}, \{C_3, p_{31}\}, \{C_4, p_{42}\}$$

in $\mathcal{C}$.

Conversely, if $\mathcal{C}$ is an exact cover of $\tau(F)$, we define a truth assigment as follows:

For every x_i, if $T_{i,\mathbf{T}}$ is in $\mathcal{C}$, then we set $v(x_i) = \mathbf{T}$, else if $T_{i,\mathbf{F}}$ is in $\mathcal{C}$, then we set $v(x_i) = \mathbf{F}$.

Example 13.3. Given the exact cover

$$T_{1,\mathbf{T}} = \{x_1, p_{21}\}, T_{2,\mathbf{T}} = \{x_2, p_{12}, p_{41}\}, T_{3,\mathbf{F}} = \{x_3, p_{23}\},\\ \{C_1, p_{11}\}, \{C_2, p_{22}\}, \{C_3, p_{31}\}, \{C_4, p_{42}\},$$

we get the satisfying assigment $v(x_1) = \mathbf{T}, v(x_2) = \mathbf{T}, v(x_3) = \mathbf{F}$.

If we now consider the proposition is CNF given by

$$F_2 = \{C_1 = (x_1 \vee \overline{x_2}),\ C_2 = (\overline{x_1} \vee x_2 \vee x_3),\ C_3 = (x_2),\\ C_4 = (\overline{x_2} \vee \overline{x_3} \vee x_4)\}$$

where we have added the boolean variable x_4 to clause C_4, then U also contains x_4 and p_{43} so we need to add the following subsets to $\mathcal{F}$:

$$T_{4,\mathbf{F}} = \{x_4, p_{43}\},\ T_{4,\mathbf{T}} = \{x_4\},\ \{C_4, p_{43}\},\ \{p_{43}\}.$$

The truth assigment $v(x_1) = \mathbf{T}, v(x_2) = \mathbf{T}, v(x_3) = \mathbf{F}, v(x_4) = \mathbf{T}$ satisfies F_2, so an exact cover $\mathcal{C}$ is

$$T_{1,\mathbf{T}} = \{x_1, p_{21}\}, T_{2,\mathbf{T}} = \{x_2, p_{12}, p_{41}\},\\ T_{3,\mathbf{F}} = \{x_3, p_{23}\}, T_{4,\mathbf{T}} = \{x_4\},\\ \{C_1, p_{11}\}, \{C_2, p_{22}\}, \{C_3, p_{31}\}, \{C_4, p_{42}\}, \{p_{43}\}.$$

Observe that this time, because the truth assignment v makes both literals corresponding to p_{42} and p_{43} true and since we picked p_{42} to form the subset $\{C_4, p_{42}\}$, we need to add the singleton $\{p_{43}\}$ to $\mathcal{C}$ to cover all elements of U.

(2) **Hamiltonian Cycle (for Directed Graphs)**

To prove that **Hamiltonian Cycle (for Directed Graphs)** is $\mathcal{NP}$-complete, we will reduce **Exact Cover** to it:

Exact Cover $\leq_P$ Hamiltonian Cycle (for Directed Graphs)

We need to find an algorithm working in polynomial time that converts an instance $(U, \mathcal{F})$ of **Exact Cover** to a directed graph $G = \tau(U, \mathcal{F})$ such that G has a Hamiltonian cycle iff $(U, \mathcal{F})$ has an exact cover.

The construction of the graph G uses a trick involving a small subgraph Gad with 7 (distinct) nodes known as a *gadget* shown in Figure 13.7.

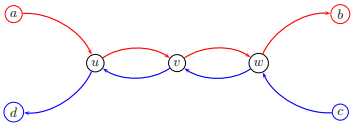


Figure 13.7: A gadget *Gad*

The crucial property of the graph *Gad* is that if *Gad* is a subgraph of a bigger graph G in such a way that no edge of G is incident to any of the nodes u, v, w unless it is one of the eight edges of *Gad* incident to the nodes u, v, w, then *for any Hamiltonian cycle in G, either the path $(a,u),(u,v),(v,w),(w,b)$ is traversed or the path $(c,w),(w,v),(v,u),(u,d)$ is traversed, but not both*.

It is convenient to use the simplified notation with a special type of edge labeled with the exclusive or sign $\oplus$ between the "edges" between a and b and between d and c, as shown in Figure 13.8.

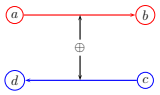


Figure 13.8: A shorthand notation for a gadget

This abbreviating device can be extended to the situation where we build gadgets between a given pair (a,b) and several other pairs $(c_1,d_1),\ldots,(c_m,d_m)$, all nodes beeing distinct, as illustrated in Figure 13.9.

Either all three edges $(c_1,d_1),(c_2,d_2),(c_3,d_3)$ are traversed or the edge (a,b) is traversed, and these possibilities are mutually exclusive.

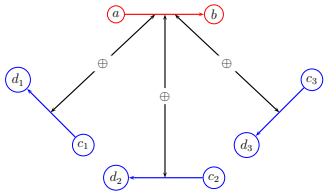


Figure 13.9: A shorthand notation for several gadgets

Example 13.4. The construction of the graph is illustrated in Figure 13.10 for the instance of the exact cover problem given by

$$U=\{u_1,u_2,u_3,u_4\},\ \mathcal{F}=\{S_1=\{u_3,u_4\},\\ S_2=\{u_2,u_3,u_4\},\ S_3=\{u_1,u_2\}\}.$$

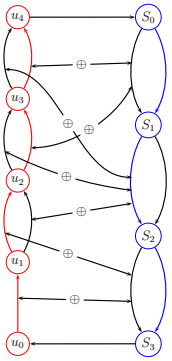


Figure 13.10: The directed graph constructed from the data $(U,\mathcal{F})$ of Example 13.4

In our example, there is a Hamiltonian where the *blue edges* are traversed between the S_i nodes, and the *red edges* are traversed between the u_j nodes.

An edge between S_i nodes which is not connected by another $\oplus$-edge is called a *short edge*, and otherwise a *long edge*.

The Hamiltonian is the following path:

short (S_0,S_1), long (S_1,S_2), short (S_2,S_3), (S_3,u_0), $(u_0,u_1)_3$, $(u_1,u_2)_3$, $(u_2,u_3)_1$, $(u_3,u_4)_1$, (u_4,S_0).

Each edge between u_{j-1} and u_j corresponds to an occurrence of u_j in some uniquely determined set $S_i\in\mathcal{F}$ (that is, $u_j\in S_i$), and we put an exclusive-or edge between the edge (u_{j-1},u_j) and the the long edge (S_{i-1},S_i) between S_{i-1} and S_i,

The subsets corresponding to the short (S_{i-1},S_i) edges are S_1 and S_3, and indeed $\mathcal{C}=\{S_1,S_3\}$ is an exact cover.

It can be proved that $(U, \mathcal{F})$ has an exact cover iff the graph $G = \tau(U, \mathcal{F})$ has a Hamiltonian cycle.

For example, if $\mathcal{C}$ is an exact cover for $(U, \mathcal{F})$, then consider the path in G obtained by traversing each *short edge* (S_{i-1}, S_i) for which $S_i \in \mathcal{C}$, each *edge* (u_{j-1}, u_j) such that $u_j \in S_i$, which means that this edge is connected by a $\oplus$-sign to the long edge (S_{i-1}, S_i) (by construction, for each u_j there is a unique such S_i), and the edges (u_n, S_0) and (S_m, u_0), then we obtain a Hamiltonian cycle.

In our example, the exact cover $\mathcal{C} = \{S_1, S_3\}$ yields the Hamiltonian

$$\text{short } (S_0, S_1),\ \text{long } (S_1, S_2),\ \text{short } (S_2, S_3),\ (S_3, u_0),$$
$$(u_0, u_1)_3,\ (u_1, u_2)_3,\ (u_2, u_3)_1,\ (u_3, u_4)_1,\ (u_4, S_0)$$

that we encountered earlier.

(3) **Hamiltonian Cycle (for Undirected Graphs)**

To show that **Hamiltonian Cycle (for Undirected Graphs)** is $\mathcal{NP}$-complete we reduce **Hamiltonian Cycle (for Directed Graphs)** to it:

Hamiltonian Cycle (for Directed Graphs) $\leq_P$ Hamiltonian Cycle (for Undirected Graphs)

Given any directed graph $G = (V, E)$ we need to construct in polynomial time an undirected graph $\tau(G) = G' = (V', E')$ such that G has a (directed) Hamiltonian cycle iff G' has a (undirected) Hamiltonian cycle.

We make three distinct copies v_0, v_1, v_2 of every node $v \in V$ which we put in V', and for every edge $(u, v) \in E$ we create five edges as illustrated in the diagram shown in Figure 13.11.

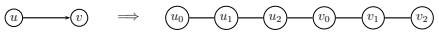


Figure 13.11: Conversion of a directed graph into an undirected graph

The crucial point about the graph G' is that although there may be several edges adjacent to a node u_0 or a node u_2, the only way to reach u_1 from u_0 is through the edge $\{u_0, u_1\}$ and the only way to reach u_1 from u_2 is through the edge $\{u_1, u_2\}$.

This implies that any Hamiltonian cycle in G' arriving to a node v_0 along an edge (u_2, v_0) must continue to node v_1 and then to v_2, which means that the edge (u, v) is traversed in G.

By considering a Hamiltonian cycle in G' or perhaps its reversal, it is not hard to show that a Hamiltonian cycle in G' determines a Hamiltonian cycle in G.

Conversely, a Hamiltonian cycle in G determines a Hamiltonian in G'.

(4) **Traveling Salesman Problem**

To show that the **Traveling Salesman Problem** is $\mathcal{NP}$-complete, we reduce the **Hamiltonian Cycle Problem (Undirected Graphs)** to it:

Hamiltonian Cycle Problem (Undirected Graphs) $\leq_P$ Traveling Salesman Problem

Given an undirected graph $G = (V, E)$, we construct an instance $\tau(G) = (D, B)$ of the traveling salesman problem so that G has a Hamiltonian cycle iff the traveling salesman problem has a solution.

If we let $n = |V|$, we have n cities and the matrix $D = (d_{ij})$ is defined as follows:

$$d_{ij} = \begin{cases} 0 & \text{if } i = j \\ 1 & \text{if } \{v_i, v_j\} \in E \\ 2 & \text{otherwise.} \end{cases}$$

We also set the budget B as $B = n$.

Any tour of the cities has cost equal to n plus the number of pairs (v_i, v_j) such that $i \neq j$ and $\{v_i, v_j\}$ is *not* an edge of G. It follows that a tour of cost n exists iff there are no pairs (v_i, v_j) of the second kind iff the tour is a Hamiltonian cycle.

(5) **Independent Set**

To show that **Independent Set** is $\mathcal{NP}$-complete, we reduce **Exact** 3**-Satisfiability** to it:

Exact 3**-Satisfiability** $\leq_P$ **Independent Set**

Recall that in **Exact** 3**-Satisfiability** every clause C_i has exactly three literals L_{i1}, L_{i2}, L_{i3}.

Given a set $F = \{C_1, \ldots, C_m\}$ of $m \geq 2$ such clauses, we construct in polynomial time an undirected graph $G = (V, E)$ such that F is satisfiable iff G has an independent set C with at least $K = m$ nodes.

For every i $(1 \leq i \leq m)$, we have three nodes c_{i1}, c_{i2}, c_{i3} corresponding to the three literals L_{i1}, L_{i2}, L_{i3} in clause C_i, so there are $3m$ nodes in V.

The "core" of G consists of m triangles, one for each set $\{c_{i1}, c_{i2}, c_{i3}\}$. We also have an edge $\{c_{ik}, c_{j\ell}\}$ iff L_{ik} and $L_{j\ell}$ are complementary literals.

Example 13.5. Let F be the set of clauses

$$F = \{C_1 = (x_1 \vee \overline{x_2} \vee x_3),\ C_2 = (\overline{x_1} \vee \overline{x_2} \vee x_3),\\ C_3 = (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}),\ C_4 = (x_1 \vee x_2 \vee x_3)\}.$$

The graph G associated with F is shown in Figure 13.12.

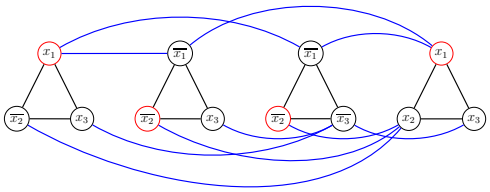


Figure 13.12: The graph constructed from the clauses of Example 13.5

Since any three nodes in a triangle are connected, an independent set C can have at most one node per triangle and thus has at most m nodes. Since the budget is $K = m$, we may assume that there is an independent with m nodes.

Define a (partial) truth assignment by

$$v(x_i) = \begin{cases} \mathbf{T} & \text{if } L_{jk} = x_i \text{ and } c_{jk} \in C \\ \mathbf{F} & \text{if } L_{jk} = \overline{x_i} \text{ and } c_{jk} \in C. \end{cases}$$

Since the non-triangle edges in G link nodes corresponding to complementary literals and nodes in C are not connected, our truth assigment does not assign clashing truth values to the variables x_i.

Not all variables may receive a truth value, in which case we assign an arbitrary truth value to the unassigned variables. This yields a satisfying assignment for F.

In Example 13.5, the set $C = \{c_{11}, c_{22}, c_{32}, c_{41}\}$ corresponding to the nodes shown in red in Figure 13.12 form an independent set, and they induce the partial truth assignment $v(x_1) = \mathbf{T}, v(x_2) = \mathbf{F}$.

The variable x_3 can be assigned an arbitrary value, say $v(x_3) = \mathbf{F}$, and v is indeed a satisfying truth assignment for F.

Conversely, if v is a truth assignment for F, then we obtain an independent set C of size m by picking for each clause C_i a node c_{ik} corresponding to a literal L_{ik} whose value under v is $\mathbf{T}$.

(6) **Clique**

To show that **Clique** is $\mathcal{NP}$-complete, we reduce **Independent Set** to it:

Independent Set $\leq_P$ Clique

The key the reduction is the notion of the complement of an undirected graph $G = (V, E)$.

The *complement* $G^c = (V, E^c)$ of the graph $G = (V, E)$ is the graph with the same set of nodes V as G but there is an edge $\{u, v\}$ (with $u \neq v$) in E^c iff $\{u, v\} \notin E$.

Then, it is not hard to check that there is a bijection between maximum independent sets in G and maximum cliques in G^c.

The reduction consists in constructing from a graph G its complement G^c, and then G has an independent set iff G^c has a clique.

This construction is illustrated in Figure 13.13, where a maximum independent set in the graph G is shown in blue and a maximum clique in the graph G^c is shown in red.

Figure 13.13: A graph (left) and its complement (right)

(7) **Node Cover**

To show that **Node Cover** is $\mathcal{NP}$-complete, we reduce **Independent Set** to it:

Independent Set $\leq_P$ Node Cover

This time the crucial observation is that if N is an independent set in G, then the complement $C = V - N$ of N in V is a node cover in G.

Thus there is an independent set of size at least K iff there is a node cover of size at most $n - K$ where $n = |V|$ is the number of nodes in V.

The reduction leaves the graph unchanged and replaces K by $n - K$.

An example is shown in Figure 13.14 where an independent set is shown in blue and a node cover is shown in red.

Figure 13.14: An inpendent set (left) and a node cover (right)

(8) **Knapsack (also called Subset sum)**

To show that **Knapsack** is $\mathcal{NP}$-complete, we reduce **Exact Cover** to it:

Exact Cover $\leq_P$ **Knapsack**

Given an instance $(U, \mathcal{F})$ of set cover with $U = \{u_1, \ldots, u_n\}$ and $\mathcal{F} = \{S_1, \ldots, S_m\}$, a family of subsets of U, we need to produce in polynomial time an instance $\tau(U, \mathcal{F})$ of the knapsack problem consisting of k nonnegative integers $a_1, \ldots, a_k$ and another integer $K > 0$ such that there is a subset $I \subseteq \{1, \ldots, k\}$ such that $\sum_{i \in I} a_i = K$ iff there is an exact cover of U using subsets in $\mathcal{F}$.

The trick here is the relationship between *set union* and *integer addition*.

Example 13.6. Consider the exact cover problem given by $U = \{u_1, u_2, u_3, u_4\}$ and

$$\mathcal{F} = \{S_1 = \{u_3, u_4\}, S_2 = \{u_2, u_3, u_4\}, S_3 = \{u_1, u_2\}\}.$$

We can represent each subset S_j by a binary string a_j of length 4, where the ith bit from the left is 1 iff $u_i \in S_j$, and 0 otherwise.

In our example

$$\begin{aligned} a_1 &= 0011 \\ a_2 &= 0111 \\ a_3 &= 1100. \end{aligned}$$

Then, the trick is that some family $\mathcal{C}$ of subsets S_j is an exact cover if the sum of the corresponding numbers a_j adds up to $1111 = 2^4 - 1 = K$.

For example,

$$\mathcal{C} = \{S_1 = \{u_3, u_4\},\ S_3 = \{u_1, u_2\}\}$$

is an exact cover and

$$a_1 + a_3 = 0011 + 1100 = 1111.$$

Unfortunately, there is a problem with this encoding which has to do with the fact that addition may involve carry. For example, assuming four subsets and the universe $U = \{u_1, \ldots, u_6\}$,

$$11 + 13 + 15 + 24 = 63,$$

in binary

$$001011 + 001101 + 001111 + 011000 = 111111,$$

but if we convert these binary strings to the corresponding subsets we get the subsets

$$\begin{aligned} S_1 &= \{u_3, u_5, u_6\} \\ S_2 &= \{u_3, u_4, u_6\} \\ S_3 &= \{u_3, u_4, u_5, u_6\} \\ S_4 &= \{u_2, u_3\}, \end{aligned}$$

which are not disjoint and do not cover U.

The fix is surprisingly simple: use *base m* (where m is the number of subsets in $\mathcal{F}$) instead of base 2.

Example 13.7. Consider the exact cover problem given by $U = \{u_1, u_2, u_3, u_4, u_5, u_6\}$ and $\mathcal{F}$ given by

$$\begin{aligned} S_1 &= \{u_3, u_5, u_6\} \\ S_2 &= \{u_3, u_4, u_6\} \\ S_3 &= \{u_3, u_4, u_5, u_6\} \\ S_4 &= \{u_2, u_3\}, \\ S_5 &= \{u_1, u_2, u_4\}. \end{aligned}$$

In base $m = 5$, the numbers corresponding to $S_1, \ldots, S_5$ are

$$\begin{aligned} a_1 &= 001011 \\ a_2 &= 001101 \\ a_3 &= 001111 \\ a_4 &= 011000 \\ a_5 &= 110100. \end{aligned}$$

This time,

$$\begin{aligned} a_1 + a_2 + a_3 + a_4 &= 001011 + 001101 + 001111 \\ &\quad + 011000 \\ &= 014223 \neq 111111, \end{aligned}$$

so $\{S_1, S_2, S_3, S_4\}$ is not a solution. However

$$a_1 + a_5 = 001011 + 110100 = 111111,$$

and $\mathcal{C} = \{S_1, S_5\}$ is an exact cover.

Thus, given an instance $(U, \mathcal{F})$ of **Exact Cover** where $U = \{u_1, \ldots, u_n\}$ and $\mathcal{F} = \{S_1, \ldots, S_m\}$ the reduction to **Knapsack** consists in forming the m numbers $a_1, \ldots, a_m$ (each of n bits) encoding the subsets S_j, namely $a_{ji} = 1$ iff $u_i \in S_j$, else 0, and to let $K = 1 + m^2 + \cdots + m^{n-1}$, which is represented in base m by the string $\underbrace{11\cdots11}_{n}$.

In testing whether $\sum_{i \in I} a_i = K$ for some subset $I \subseteq \{1, \ldots, m\}$, we use *arithmetic in base m*.

If a candidate solution $\mathcal{C}$ involves at most $m-1$ subsets, then since the corresponding numbers are added in base m, a carry can never happen.

If the candidate solution involves all m subsets, then $a_1 + \cdots + a_m = K$ iff $\mathcal{F}$ is a partition of U, since otherwise some bit in the result of adding up these m numbers in base m is not equal to 1, even if a carry occurs.

(9) **Inequivalence of $*$-free Regular Expressions**

To show that **Inequivalence of $*$-free Regular Expressions** is $\mathcal{NP}$-complete, we reduce the **Satisfiability Problem** to it:

Satisfiability Problem $\leq_P$ Inequivalence of $*$-free Regular Expressions

We already argued that **Inequivalence of $*$-free Regular Expressions** is in $\mathcal{NP}$ because if R is a $*$-free regular expression, then for every string $w \in \mathcal{L}[R]$ we have $|w| \leq |R|$.

We reduce the **Satisfiability Problem** to the **Inequivalence of $*$-free Regular Expressions** as follows.

For any set of clauses $P = C_1 \wedge \cdots \wedge C_p$, if the propositional variables occurring in P are $x_1, \ldots, x_n$, we produce two $*$-free regular expressions R, S over $\Sigma = \{0, 1\}$, such that P is satisfiable iff $L_R \neq L_S$.

The expression S is actually

$$S = \underbrace{(0+1)(0+1)\cdots(0+1)}_{n}.$$

The expression R is of the form

$$R = R_1 + \cdots + R_p,$$

where R_i is constructed from the clause C_i in such a way that L_{R_i} corresponds precisely to the set of truth assignments that falsify C_i; see below.

Given any clause C_i, let R_i be the $*$-free regular expression defined such that, if x_j and $\overline{x}_j$ both belong to C_i (for some j), then $R_i = \emptyset$, else

$$R_i = R_i^1 \cdot R_i^2 \cdots R_i^n,$$

where R_i^j is defined by

$$R_i^j = \begin{cases} 0 & \text{if } x_j \text{ is a literal of } C_i \\ 1 & \text{if } \overline{x}_j \text{ is a literal of } C_i \\ (0+1) & \text{if } x_j \text{ does not occur in } C_i. \end{cases}$$

For example, to express the fact that every position is tiled by a single tile, use the equation

$$\sum_{t \in \mathcal{T}} x_{mnt} = 1,$$

for all m, n with $1 \leq m \leq 2s$ and $1 \leq n \leq s$. We leave the rest as as exercise.

(10) 0-1 **integer programming problem**

It is easy to check that the problem is in $\mathcal{NP}$.

To prove that the is $\mathcal{NP}$-complete we reduce the **bounded-tiling problem** to it:

bounded-tiling problem $\leq_P$ 0-1 **integer programming problem**

Given a tiling problem, $((\mathcal{T}, V, H), \widehat{s}, \sigma_0)$, we create a 0-1-valued variable x_{mnt}, such that $x_{mnt} = 1$ iff tile t occurs in position (m, n) in some tiling.

Write equations or inequalities expressing that a tiling exists and then use "slack variables" to convert inequalities to equations.

13.3 Succinct Certificates, co$\mathcal{NP}$, and $\mathcal{EXP}$

All the problems considered in Section 13.1 share a common feature, which is that for each problem, *a solution is produced nondeterministically* (an exact cover, a directed Hamiltonian cycle, a tour of cities, an independent set, a node cover, a clique *etc.*), and then *this candidate solution is checked deterministically and in polynomial time*. The candidate solution is a string called a *certificate* (or *witness*).

It turns out that membership on $\mathcal{NP}$ can be defined in terms of certificates.

To be a certificate, a string must satisfy two conditions:

1. It must be *polynomially succinct*, which means that its length is at most a polynomial in the length of the input.
2. It must be *checkable* in polynomial time.

All "yes" inputs to a problem in $\mathcal{NP}$ must have at least one certificate, while all "no" inputs must have none.

The notion of certificate can be formalized using the notion of a polynomially balanced language.

Definition 13.3. Let Σ be an alphabet, and let ";" be a symbol not in Σ. A language $L' \subseteq \Sigma^*; \Sigma^*$ is said to be *polynomially balanced* if there exists a polynomial $p(X)$ such that for all $x, y \in \Sigma^*$, if $x; y \in L'$ then $|y| \leq p(|x|)$.

Suppose L' is a polynomially balanced language and that $L' \in \mathcal{P}$. Then we can consider the language

$$L = \{x \in \Sigma^* \mid (\exists y \in \Sigma^*)(x; y \in L')\}.$$

The intuition is that for each $x \in L$, the set

$$\{y \in \Sigma^* \mid x; y \in L'\}$$

is the set of certificates of x.

For every $x \in L$, a Turing machine can nondeterministically guess one of its certificates y, and then use the deterministic Turing machine for L' to check in polynomial time that $x; y \in L'$. It follows that $L \in \mathcal{NP}$.

Conversely, if $L \in \mathcal{NP}$ and the alphabet Σ has at least two symbols, we can encode the paths in the computation tree for every input $x \in L$, and we obtain a polynomially balanced language $L' \subseteq \Sigma^*; \Sigma^*$ in $\mathcal{P}$ such that

$$L = \{x \in \Sigma^* \mid (\exists y \in \Sigma^*)(x; y \in L')\}.$$

In summary, we obtain the following theorem.

Theorem 13.1. *Let $L \subseteq \Sigma^*$ be a language over an alphabet Σ with at least two symbols, and let ";" be a symbol not in Σ. Then $L \in \mathcal{NP}$ iff there is a polynomially balanced language $L' \subseteq \Sigma^*; \Sigma^*$ such that $L' \in \mathcal{P}$ and*

$$L = \{x \in \Sigma^* \mid (\exists y \in \Sigma^*)(x; y \in L')\}.$$

A striking illustration of the notion of succint certificate is illustrated by the set of *composite* integers, namely those natural numbers $n \in \mathbb{N}$ that can be written as the product pq of two numbers $p, q \geq 2$ with $p, q \in \mathbb{N}$.

For example, the number

$$4,294,967,297$$

is a composite!

This is far from obvious, but if an oracle gives us the certificate $\{6,700,417,\ \ 641\}$, it is easy to carry out in polynomial time the multiplication of these two numbers and check that it is equal to $4,294,967,297$.

Finding a certificate is usually (very) hard, but checking that it works is easy. This is the point of certificates.

We conclude this section with a brief discussion of the complexity classes co$\mathcal{NP}$ and $\mathcal{EXP}$.

By definition,

$$\text{co}\mathcal{NP} = \{\overline{L} \mid L \in \mathcal{NP}\},$$

that is, co$\mathcal{NP}$ consists of all complements of languages in $\mathcal{NP}$.

Since $\mathcal{P} \subseteq \mathcal{NP}$ and $\mathcal{P}$ is closed under complementation,

$$\mathcal{P} \subseteq \text{co}\mathcal{NP},$$

but nobody knows whether $\mathcal{NP}$ is closed under complementation, that is, nobody knows whether $\mathcal{NP} = \text{co}\mathcal{NP}$.

What can be shown is that if $\mathcal{NP} \neq \text{co}\mathcal{NP}$ then $\mathcal{P} \neq \mathcal{NP}$.

However it is possible that $\mathcal{P} \neq \mathcal{NP}$ and yet $\mathcal{NP} = \text{co}\mathcal{NP}$, although this is considered unlikely.

Of course, $\mathcal{P} \subseteq \mathcal{NP} \cap \text{co}\mathcal{NP}$.

There are problems in $\mathcal{NP} \cap \text{co}\mathcal{NP}$ not known to be in $\mathcal{P}$. One of the most famous in the following problem:

Integer factorization problem:

Given an integer $N \geq 3$, and another integer M (a budget) such that $1 < M < N$, does N have a factor d with $1 < d \leq M$?

That **Integer factorization** is in $\mathcal{NP}$ is clear.

To show that **Integer factorization** is in co$\mathcal{NP}$, we can guess a factorization of N into distinct factors all greater than M, check that they are prime using the results of Chapter **??** showing that testing primality is in $\mathcal{NP}$ (even in $\mathcal{P}$, but that's much harder to prove), and then check that the product of these factors is N.

It is widely believed that **Integer factorization** does not belong to $\mathcal{P}$, which is the technical justification for saying that this problem is hard.

Most cryptographic algorithms rely on this unproven fact.

If **Integer factorization** was either $\mathcal{NP}$-complete or co$\mathcal{NP}$-complete, then we would have $\mathcal{NP} = \text{co}\mathcal{NP}$, which is considered very unlikely.

Remark: If $\sqrt{N} \leq M < N$, the above problem is equivalent to asking whether N is prime.

A natural instance of a problem in co$\mathcal{NP}$ is the *unsatisfiability problem* for propositions, namely deciding that a proposition P has no satisfying assignmnent.

A proposition P (in CNF) is *falsifiable* if there is some truth assigment v such that $\widehat{v}(P) = \mathbf{F}$.

It is obvious that the set of falsifiable propositions is in $\mathcal{NP}$. Since a proposition P is valid iff P is not falsifiable, the *validity (or tautology) problem* TAUT for propositions is in co$\mathcal{NP}$.

In fact, TAUT is co$\mathcal{NP}$-complete.

Despite the fact that this problem has been extensively studied, not much is known about its exact complexity.

The reasoning used to show that TAUT is co$\mathcal{NP}$-complete can also be used to show the following interesting result.

Proposition 13.2. *If a language L is $\mathcal{NP}$-complete, then its complement $\overline{L}$ is co$\mathcal{NP}$-complete.*

The class $\mathcal{EXP}$ is defined as follows.

Definition 13.4. A deterministic Turing machine M is said to be *exponentially bounded* if there is a polynomial $p(X)$ such that for every input $x \in \Sigma^*$, there is no ID ID_n such that

$$ID_0 \vdash ID_1 \vdash^* ID_{n-1} \vdash ID_n, \quad \text{with} \quad n > 2^{p(|x|)}.$$

The class $\mathcal{EXP}$ is the class of all languages that are accepted by some exponentially bounded deterministic Turing machine.

Remark: We can also define the class $\mathcal{NEXP}$ as in Definition 13.4, except that we allow nondeterministic Turing machines.

One of the interesting features of $\mathcal{EXP}$ is that it contains $\mathcal{NP}$.

Theorem 13.3. *We have the inclusion $\mathcal{NP} \subseteq \mathcal{EXP}$.*

It is also immediate to see that $\mathcal{EXP}$ is closed under complementation. Furthermore the strict inclusion $\mathcal{P} \subset \mathcal{EXP}$ holds.

Theorem 13.4. *We have the strict inclusion $\mathcal{P} \subset \mathcal{EXP}$.*

The proof involves a diagonalization argument to produce a language E such that $E \notin \mathcal{P}$, yet $E \in \mathcal{EXP}$.

In summary, we have the chain of inclusions

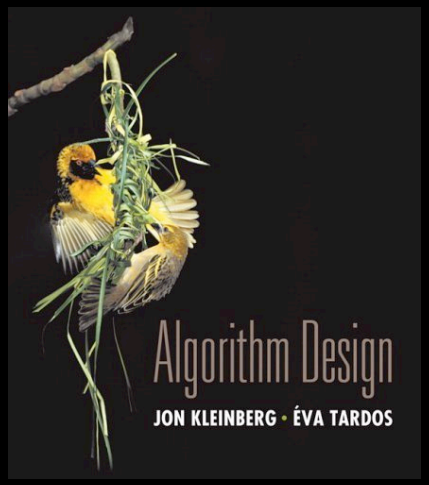
$$\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{EXP},$$

where the left inclusion and the right inclusion are both open problems, but we know that at least one of these two inclusions is strict.

We also have the inclusions

$$\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{EXP} \subseteq \mathcal{NEXP}.$$

Nobody knows whether $\mathcal{EXP} = \mathcal{NEXP}$, but it can be shown that if $\mathcal{EXP} \neq \mathcal{NEXP}$, then $\mathcal{P} \neq \mathcal{NP}$; see Papadimitriou [**?**].



Chapter 8

NP and Computational Intractability

Algorithm Design Patterns and Anti-Patterns

Algorithm design patterns.	Ex.
Greedy.	$O(n \log n)$ interval scheduling.
Divide-and-conquer.	$O(n \log n)$ FFT.
Dynamic programming.	$O(n^2)$ edit distance.
Duality.	$O(n^3)$ bipartite matching.
Reductions.	
Local search.	
Randomization.	

Algorithm design anti-patterns.	
NP-completeness.	$O(n^k)$ algorithm unlikely.
PSPACE-completeness.	$O(n^k)$ certification algorithm unlikely.
Undecidability.	No algorithm possible.

8.1 Polynomial-Time Reductions

Classify Problems According to Computational Requirements

Q. Which problems will we be able to solve in practice?

A working definition. [von Neumann 1953, Godel 1956, Cobham 1964, Edmonds 1965, Rabin 1966]
Those with polynomial-time algorithms.

Yes	Probably no
Shortest path	Longest path
Matching	3D-matching
Min cut	Max cut
2-SAT	3-SAT
Planar 4-color	Planar 3-color
Bipartite vertex cover	Vertex cover
Primality testing	Factoring

Classify Problems

Desiderata. Classify problems according to those that can be solved in polynomial-time and those that cannot.

Provably requires exponential-time.

- Given a Turing machine, does it halt in at most k steps?
- Given a board position in an n-by-n generalization of chess, can black guarantee a win?

Frustrating news. Huge number of fundamental problems have defied classification for decades.

This chapter. Show that these fundamental problems are "computationally equivalent" and appear to be different manifestations of one really hard problem.

Polynomial-Time Reduction

Desiderata'. Suppose we could solve X in polynomial-time. What else could we solve in polynomial time?

don't confuse with reduces from

Reduction. Problem X polynomial reduces to problem Y if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to oracle that solves problem Y.

computational model supplemented by special piece of hardware that solves instances of Y in a single step

Notation. $X \le_P Y$.

Remarks.

- We pay for time to write down instances sent to black box $\Rightarrow$ instances of Y must be of polynomial size.
- Note: Cook reducibility.

in contrast to Karp reductions

Polynomial-Time Reduction

Purpose. Classify problems according to relative difficulty.

Design algorithms. If $X \le_P Y$ and Y can be solved in polynomial-time, then X can also be solved in polynomial time.

Establish intractability. If $X \le_P Y$ and X cannot be solved in polynomial-time, then Y cannot be solved in polynomial time.

Establish equivalence. If $X \le_P Y$ and $Y \le_P X$, we use notation $X \equiv_P Y$.

up to cost of reduction

Reduction By Simple Equivalence

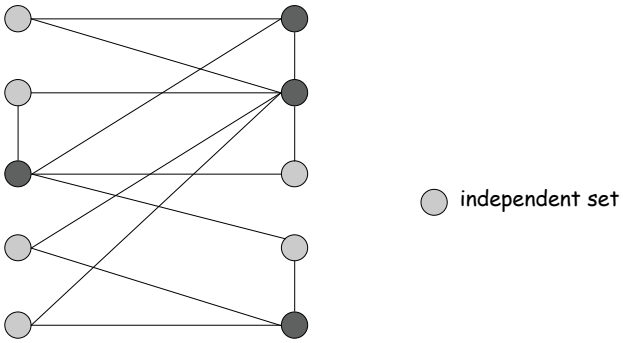
Basic reduction strategies.

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

Independent Set

INDEPENDENT SET: Given a graph $G = (V, E)$ and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and for each edge at most one of its endpoints is in S?

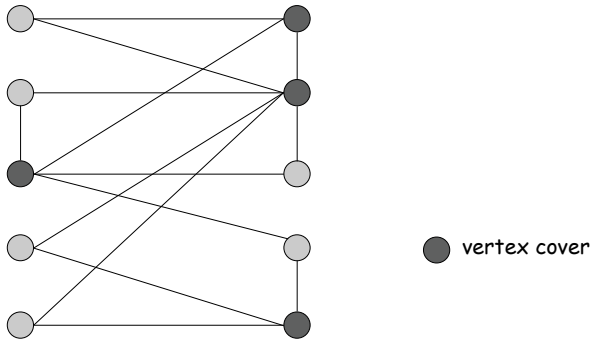
Ex. Is there an independent set of size ≥ 6? Yes.
Ex. Is there an independent set of size ≥ 7? No.



Vertex Cover

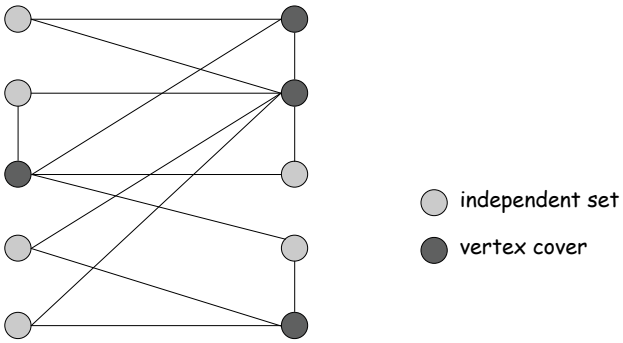
VERTEX COVER: Given a graph $G = (V, E)$ and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge, at least one of its endpoints is in S?

Ex. Is there a vertex cover of size ≤ 4? Yes.
Ex. Is there a vertex cover of size ≤ 3? No.



Vertex Cover and Independent Set

Claim. VERTEX-COVER $\equiv_P$ INDEPENDENT-SET.
Pf. We show S is an independent set iff $V - S$ is a vertex cover.



Vertex Cover and Independent Set

Claim. VERTEX-COVER $\equiv_P$ INDEPENDENT-SET.
Pf. We show S is an independent set iff $V - S$ is a vertex cover.

$\Rightarrow$

- Let S be any independent set.
- Consider an arbitrary edge (u, v).
- S independent $\Rightarrow u \notin S$ or $v \notin S \Rightarrow u \in V - S$ or $v \in V - S$.
- Thus, $V - S$ covers (u, v).

$\Leftarrow$

- Let $V - S$ be any vertex cover.
- Consider two nodes $u \in S$ and $v \in S$.
- Observe that $(u, v) \notin E$ since $V - S$ is a vertex cover.
- Thus, no two nodes in S are joined by an edge $\Rightarrow S$ independent set. ▪

Reduction from Special Case to General Case

Basic reduction strategies.

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

SET COVER: Given a set U of elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of U, and an integer k, does there exist a collection of ≤ k of these sets whose union is equal to U?

Sample application.

- m available pieces of software.
- Set U of n capabilities that we would like our system to have.
- The ith piece of software provides the set $S_i \subseteq U$ of capabilities.
- Goal: achieve all n capabilities using fewest pieces of software.

Ex:

U = { 1, 2, 3, 4, 5, 6, 7 }
k = 2

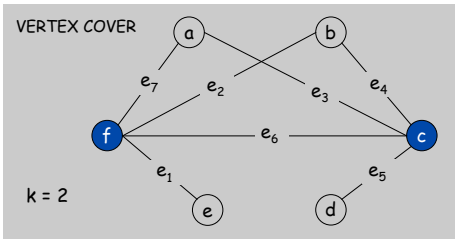
S_1 = {3, 7}	S_4 = {2, 4}
S_2 = {3, 4, 5, 6}	S_5 = {5}
S_3 = {1}	S_6 = {1, 2, 6, 7}

Vertex Cover Reduces to Set Cover

Claim. VERTEX-COVER ≤ $_P$ SET-COVER.
Pf. Given a VERTEX-COVER instance G = (V, E), k, we construct a set cover instance whose size equals the size of the vertex cover instance.

Construction.

- Create SET-COVER instance:
  - k = k, U = E, S_v = {e ∈ E : e incident to v }
- Set-cover of size ≤ k iff vertex cover of size ≤ k. ▪



SET COVER

U = { 1, 2, 3, 4, 5, 6, 7 }
k = 2

S_a = {3, 7}	S_b = {2, 4}
S_c = {3, 4, 5, 6}	S_d = {5}
S_e = {1}	S_f = {1, 2, 6, 7}

Polynomial-Time Reduction

Basic strategies.

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

8.2 Reductions via "Gadgets"

Basic reduction strategies.

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction via "gadgets."

Literal: A Boolean variable or its negation. x_i or $\overline{x_i}$

Clause: A disjunction of literals. $C_j = x_1 \vee \overline{x_2} \vee x_3$

Conjunctive normal form: A propositional formula Φ that is the conjunction of clauses. $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

SAT: Given CNF formula Φ, does it have a satisfying truth assignment?

3-SAT: SAT where each clause contains exactly 3 literals.

each corresponds to a different variable

Ex: $(\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$

Yes: x_1 = true, x_2 = true x_3 = false.

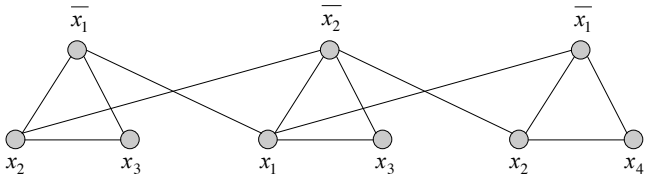
Claim. 3-SAT $\leq_P$ INDEPENDENT-SET.

Pf. Given an instance Φ of 3-SAT, we construct an instance (G, k) of INDEPENDENT-SET that has an independent set of size k iff Φ is satisfiable.

Construction.

- G contains 3 vertices for each clause, one for each literal.
- Connect 3 literals in a clause in a triangle.
- Connect literal to each of its negations.

G



k = 3

$\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$

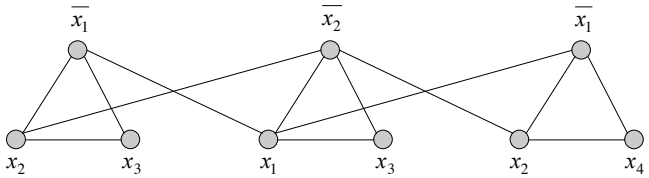
Claim. G contains independent set of size k = $|\Phi|$ iff Φ is satisfiable.

Pf. $\Rightarrow$ Let S be independent set of size k.

- S must contain exactly one vertex in each triangle.
- Set these literals to true. ← and any other variables in a consistent way
- Truth assignment is consistent and all clauses are satisfied.

Pf $\Leftarrow$ Given satisfying assignment, select one true literal from each triangle. This is an independent set of size k. ▪

G



k = 3

$\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$

Review

Basic reduction strategies.

- Simple equivalence: INDEPENDENT-SET $\equiv_P$ VERTEX-COVER.
- Special case to general case: VERTEX-COVER $\leq_P$ SET-COVER.
- Encoding with gadgets: 3-SAT $\leq_P$ INDEPENDENT-SET.

Transitivity. If $X \leq_P Y$ and $Y \leq_P Z$, then $X \leq_P Z$.
Pf idea. Compose the two algorithms.

Ex: 3-SAT $\leq_P$ INDEPENDENT-SET $\leq_P$ VERTEX-COVER $\leq_P$ SET-COVER.

Self-Reducibility

Decision problem. Does there exist a vertex cover of size $\leq$ k?
Search problem. Find vertex cover of minimum cardinality.

Self-reducibility. Search problem $\leq_P$ decision version.

- Applies to all (NP-complete) problems in this chapter.
- Justifies our focus on decision problems.

Ex: to find min cardinality vertex cover.

- (Binary) search for cardinality k* of min vertex cover.
- Find a vertex v such that $G - \{ v \}$ has a vertex cover of size $\leq$ k* - 1.
  - any vertex in any min vertex cover will have this property
- Include v in the vertex cover.
- Recursively find a min vertex cover in $G - \{ v \}$.

delete v and all incident edges