§6.4. The class $\mathcal{NP}$ defined

Our **aspiration**: We want to identify problems that are *not* in class $\mathcal{P}$.

We *suspect* Ham-Cycle, TSP, Indep-Set, Partition, SAT, and 3-SAT are *not* in class $\mathcal{P}$. They all happen to be in class $\mathcal{NP}$.

A language L is in class $\mathcal{NP}$ if there exists a nondeterministic Turing machine M such that

- All computations are bounded by a polynomial in the size of the input (and hence halt)
- There are no false positives:

  If $w \notin L$ then all computations of M on w halt n
- There may be some false negatives, but there must be at least one true positive
  If $w \in L$, then $\exists$ a computation of M on w that halts y

LP pg 293

Notice how cleverly the nodeterministic "algorithms" of [Examples 6.4.(1&2)] exploit the *fundamental asymmetry* in the definition of nondeterministic time-bounded computation. They try out all possible solutions to the problem in hand in independent computations, and accept as soon as they discover one that works—oblivious of the others that do not. LP pg 295

- $\mathcal{P} \subseteq \mathcal{NP}$, just as $R \subseteq RE$.
- $\mathcal{P} \subseteq \mathcal{EXP}$, but $\mathcal{P} \neq \mathcal{EXP}$.
  (since $E \in \mathcal{EXP}$ but $E \notin \mathcal{P}$, Theorem 6.1.2)
- $\mathcal{NP} \subseteq \mathcal{EXP}$. (Theorem 6.4.1)
- These imply that $\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{EXP}$,
  but also that $\mathcal{P} = \mathcal{NP}$ and $\mathcal{NP} = \mathcal{EXP}$ cannot *both* be true.
- We don't know whether $\mathcal{P} \neq \mathcal{NP}$ or $\mathcal{NP} \neq \mathcal{EXP}$ (possibly both are true).

Alternative definition of $\mathcal{NP}$:

> $L \in \mathcal{NP}$ if there exists a Turing machine M such that for all $w \in L$ there exists a string y such that $|y|$ is polynomial in $|w|$ and M computes whether $w \in L$ in polynomial time when given $w; y$ as input.

y is a **succinct certificate**.

CLRS's definition of class $\mathcal{NP}$:

> *The* **complexity class** $\mathcal{NP}$ *is the class of languages that can be verified by a polynomial-time algorithm. More precisely, a language L belongs to $\mathcal{NP}$ if and only if there exist a two-input polynomial-time algorithm A and a constant c such that*
>
> $$L = \{x \in \{0,1\}* \mid \begin{array}{l} \exists \textit{ a certificate } y \textit{ with } |y| = O(|x|^c) \\ \textit{such that } A(x,y) = 1\} \end{array}$$
>
> *We say that algorithm A* **verifies** *language L* **in polynomial time**. *CLRS pg 1064*

Revisiting the **nature of a reduction**:

- A reduction from A to B uses a solution to B to build a solution to A. "If we can solve B [within constraints], then we can solve A [within analogous constraints]."
- To show a polynomial reduction from L_1 to L_2 requires us to
  - Describe a function τ from L_1-candidates to L_2-candidates
  - Show that τ is computed in polynomial time.
  - Show that $\forall\ x \in L_1$-candidates, $x \in L_1$ iff $\tau(x) \in L_2$.

  So the reduction turns an instance of "problem" L_1 to an instance of "problem" L_2.
- A reduction from A to B is evidence that B is at least as hard as A.

We need to show there exists a Hamiltonian cycle in $G = (V, E)$ iff there exists a satisfying truth assignment to the formula.

Proof *($\Rightarrow$) Suppose T satisfies the formula. Then for each $v_i \in V$ (that is, each $i \in [1, n]$), exactly one x_{ij} is true. For each $j \in [1, n]$ (that is, for each position in the cycle), exactly one x_{ij} is true. If x_{ij} and x_{kj+1} are both true, then $(v_i, v_k) \in E$.*

($\Leftarrow$) Conversely, suppose there exists a Hamiltonian cycle for G. Then the truth assignment T where $T(x_{ij}) = \top$ iff v_i is in the jth position in the cycle satisfies the formula. $\square$

This is **bad news** for SAT.

- If we could solve SAT in polynomial time [or any other time category], then we could solve HamCycle in polynomial time [or whatever category]
- If we prove HamCycle can't be solved in polynomial time, then SAT also can't.
- If we prove SAT can't be done in polynomial time, then the story still isn't over for HamCycle.

Example 7.1.2: Reducing Knapsack to Partition

Knapsack: Given a set S of n integers and capacity k, is there [find] a subset of S that sum exactly to k?
Partition: Given a set S of n integers, can they be partitioned exactly in half (in terms of their sum)?

Let $S = \{a_1, a_2, \ldots a_n\}, k$ be an instance of Knapsack.

Let $H = \frac{1}{2}\sum_{a_i \in S} a_i$ and make set $S_2 = S \cup \{2H + 2k, 4H\}$. This is an instance of Partition.
Suppose a partition exists for S_2, call it $P \cup \{4H\}$ and $(S - P) \cup \{2H + 2k\}$ for some $P \subseteq S$. Then

$$\begin{aligned} 4H + \sum_{a_i \in P} a_i &= 2H + 2k + \sum_{a_i \in S-P} a_i \\ 4H + 2\sum_{a_i \in P} a_i &= 2H + 2k + \sum_{a_i \in S} a_i = 2H + 2k + 2H = 4H + 2k \\ \sum_{a_i \in P} a_i &= k \end{aligned}$$

And so P is our solution to Knapsack.

Conversely, suppose there exists $P \subseteq S$, a solution to Knapsack, that is, $\sum_{a_i \in P} a_i = k$.
Work backwards algebraically ...

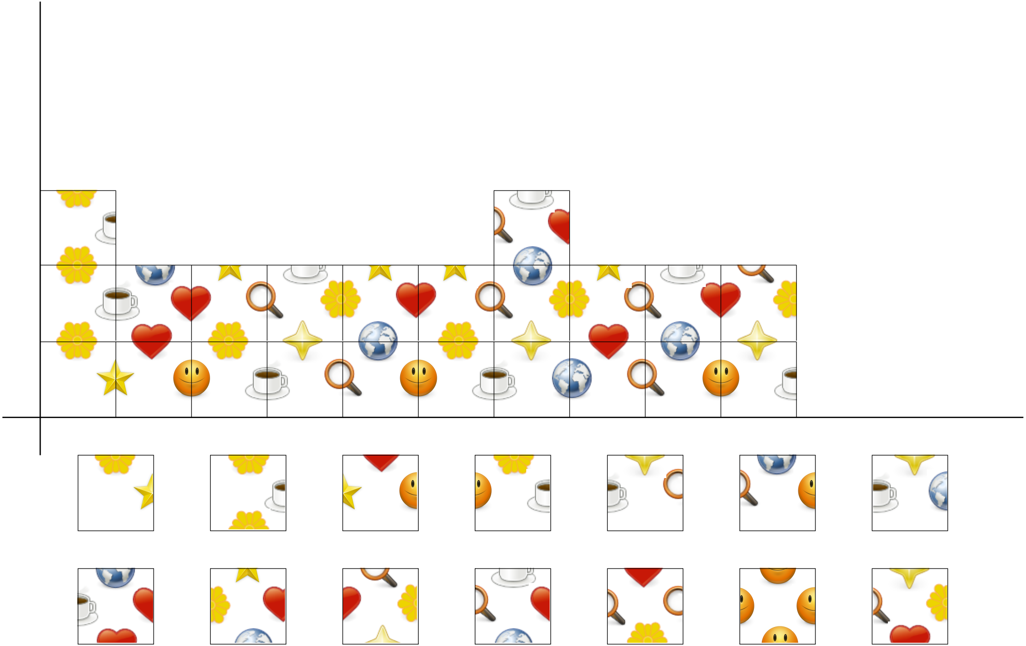
Definition 7.1.2.: A language $L \subseteq \Sigma*$ is $\mathcal{NP}$-**complete** if

1. $L \in \mathcal{NP}$
2. For every language $L' \in \mathcal{NP}$, there is a polynomial reduction from L' to L [L is $\mathcal{NP}$-**hard**].

Let $\mathcal{NPC}$ be the class of $\mathcal{NP}$-complete languages.

Theorem 7.1.1: $\mathcal{P} = \mathcal{NP}$ iff $\exists\ L \in \mathcal{NPC}$ such that $L \in \mathcal{P}$.

Proving that a problem is $\mathcal{NP}$-complete shows that it is at least as hard as all the other problems shown to be $\mathcal{NP}$-complete.



Bounded tiling: Like the original tiling problem, but we are given the entire first row, and we need to tile only a certain portion, an $s \times s$ square.

The $\mathcal{NP}$-completeness proof:

Bounded-Tiling is in class $\mathcal{NP}$: The certificate is the $s \times s$ square. We can check that the square is legal in $O(s^2)$ time. This is polynomial in the size of the input, since the size of the input is $\Omega(s)$.

Now, suppose $L \in \mathcal{NP}$. Then there exists M, a nondeterministic Turing machine that decides L in $p(|x|)$ for some polynomial p, where x ranges over the candidate strings for L.

(Very informal:) Base s on $p(|x|)$, and set up a tiling system analogous to the proof that the original tiling problem is undecidable. A tiling exists iff a computation that accepts x exists (and hence $x \in L$). $\square$

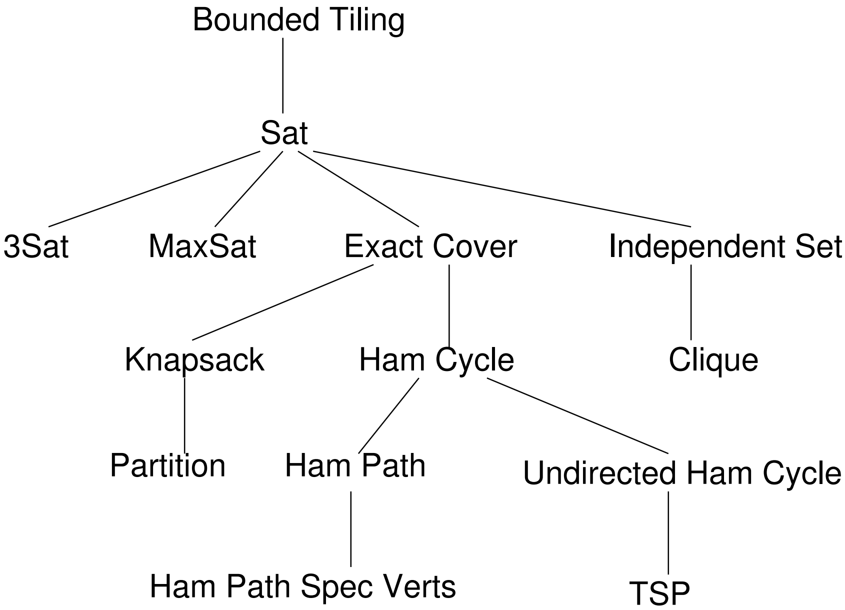
A. Prove $L \in \mathcal{NP}$

1. Describe a certificate.
2. Demonstrate that the certificate can be used to check a string/solution in polynomial time.
3. Demonstrate that the certificate itself is succinct (polynomial in size)

usually easy for our problems—ok to do briefly/informally

B. Prove L is $\mathcal{NP}$-hard

1. Choose a known $\mathcal{NP}$-complete problem L_2.
2. Describe a reduction τ from L_2 to L.
3. Demonstrate τ can be computed in polynomial time. (Also usually easy.)
4. Demonstrate that $x \in L_2$ iff $\tau(x) \in L$


Bounded Tiling
Sat
3Sat
MaxSat
Exact Cover
Independent Set
Knapsack
Ham Cycle
Clique
Partition
Ham Path
Undirected Ham Cycle
Ham Path Spec Verts
TSP

Reducing Sat to Exact Cover:
Suppose $\{c_1, c_2, \dots c_\ell\}$ is an instance of Sat.
Define the following instance of Exact Cover:

$$\begin{array}{rcll} & & \{x_i\} & \text{for each variable } i \\ \mathcal{U} = & \cup & \{c_j\} & \text{for each clause } j \\ & \cup & \{p_{jk}\} & \text{for each position } k \text{ in clause } j \end{array}$$

$$\mathcal{F} = \begin{array}{lll} \{ & \forall j,k & \{p_{jk}\} \\ & \forall i & T_{i\top} = \{x_i\} \cup \{p_{jk} \mid \lambda_{jk} = \sim x_i\} \\ & \forall i & T_{i\bot} = \{x_i\} \cup \{p_{jk} \mid \lambda_{jk} = x_i\} \\ & \forall j,k & \{c_j p_{jk}\} \qquad\qquad\qquad\qquad \} \end{array}$$

- At least one of $T_{i\bot}$ or $T_{i\top}$ for each i must be in the cover, which stands for the truth assignment.
- At least one of $\{c_j p_{jk}\}$ must be in the cover, which stands for which literal satisfies clause j.
- The extra $\{p_{jk}\}$ sets can be chosen as necessary to account for literals not used in satisfying the formula.

Proof that HAMILTONPATH is $\mathcal{NP}$-Complete

Proof. *[HAMILTONPATH is $\mathcal{NP}$.] Suppose $G = (V, E)$ is a graph, an instance of the HAMILTONPATH. Let $p = \langle u_1, u_2, \ldots u_n \rangle$ be a a sequence of vertices from V, a proposed Hamilton path in G. With any reasonable representation of G, one can check that each vertex in V appears uniquely in p, and that for any pair of vertices u_i, u_{i+1} as they appear in p, the edge (u_i, u_{i+1}) is in E. Moreover, the path p is smaller than the representation of G, so it is succinct.*

[HAMILTONPATH is $\mathcal{NP}$-hard.] Next, suppose $G = (E, V)$ is an instance of HAMILTONCYCLE. Let $v_1 \in V$ be an arbitrary vertex. Let $G' = (V', E')$ be a new graph such that v_1 is removed and four new vertices are added, that is, $V' = V - \{v_1\} \cup \{v_a, v_b, v_c, v_d\}$; and every edge that is incident on v_1 is replaced with two analogous edges incident on v_b and v_c, and and edges (v_a, v_b) and (v_c, v_d) are added, that is

$$
\begin{aligned}
E' \quad = \quad & (E - \{(v_1, v_x) \mid (v_1, v_x) \in E\}) \\
& \cup \{(v_b, v_x), (v_c, v_x) \mid (v_1, v_x) \in E\} \\
& \cup \{(v_a, v_b), (v_c, v_d)\}
\end{aligned}
$$

This reduction reduction is accomplished by one pass over the edges, which is polynomially computable.
Now, suppose G has a Hamilton cycle, call it $(v_1, v_2, \ldots v_{|V|-1}, v_1)$. (As a cycle, it has an arbitrary starting/ending point, so we are free to choose v_1 as the starting point when naming the cycle.) Then G' has a Hamiltonian path $(v_a, v_b, v_2, \ldots, v_{|V|-1}, v_c, v_d)$.
Conversely, suppose G' has a Hamiltonian path. Based on how we constructed G' (for example, the only edge going out of v_a is (v_a, v_b), and the only edge going into v_d is (v_c, v_d)), that path must be in the form $(v_a, v_b, v_2, \ldots, v_{|V|-1}, v_c, v_d)$. Then G has a Hamiltonian cycle $(v_1, v_2, \ldots v_{|V|-1}, v_1)$.
Therefore Hamilton Path *is* $\mathcal{NP}$*-complete.* $\square$

Proof that Longest Cycle is $\mathcal{NP}$-Complete

Proof. *[Longest Cycle is $\mathcal{NP}$.] Suppose $(G = (V, E), K)$ is an instance of Longest Cycle and p is a path that is a proposed cycle of length K. An algorithm to check that p is consistent with E, has no repeated vertices, and has length at least K, is polynomial with any reasonable representation of G. Moreover, since p is no larger than the representation of G, it is succinct.*
[Longest Cycle is $\mathcal{NP}$-hard.] Suppose $(G = (V, E))$ is an instance of Hamilton Cycle. Then make an instance of Longest Cycle by letting $K = |V|$, which obviously can be done in polynomial time.
Since $K = |V|$, any cycle of length (at least) K must be a Hamilton cycle, and any Hamilton cycle must have length K.
Therefore Longest Cycle is $\mathcal{NP}$-complete. $\square$

Proof that SUBGRAPH ISOMORPHISM is $\mathcal{NP}$-Complete

Proof. *[SUBGRAPH ISOMORPHISM is $\mathcal{NP}$.] Suppose $(G_1 = (V_1, E_1), G_2 = (V_2, E_2))$ is an instance of SUBGRAPH ISOMORPHISM and f is a function $V_1 \to V_2$ (expressed as a list of pairs where $(v_{1,a}, v_{2,b})$ indicates $v_{1,a} \in V_1$, $v_{2,b} \in V_2$, and $f(v_{1,a}) = v_{2,b}$) proposed as an isomorphism. An algorithm to check that f is a one-to-one function and that for all $(v_{1,a}, v_{1,b}) \in E_1$, $(f(v_{1,a}), f(v_{1,b})) \in E_2$, is polynomial with any reasonable representation of G. Moreover, since $|f| = O(V_1)$, it is succinct.*

[SUBGRAPH ISOMORPHISM is $\mathcal{NP}$-hard.] Suppose $(H = (W, F))$ is an instance of HAMILTON CYCLE. Then construct a graph $G = (V, E)$ that such that $|V| = |W|$ and $E = \{(w_1, w_2), (w_2, w_3), \ldots (w_{|V|}, w_1)\}$ An algorithm to construct this graph takes $O(V)$ time.

Note that E has only those edges that make a Hamiltonian cycle. Thus G is isomorphic to a subgraph of H iff H has a Hamiltonian cycle.

Therefore SUBGRAPH ISOMORPHISM is $\mathcal{NP}$-complete. $\square$

Reduction from UHC to TSP (LP pg 324).

Differences between UHC and TSP:

- The graph in TSP is *weighted* (interpreted as distances)
- The graph in TSP is *complete*
- A TSP problem has a *budget*

Suppose we have an instance of UHC, an undirected graph $G = (V, E)$. Construct a graph with the same vertices but complete in its edges and with distances

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

Set the budget to $|V|$.

Reduction from Exact Cover to Knapsack (LP pg 325).

Given an instance of Exact Cover $(\mathcal{U}, \mathcal{F} \subseteq \mathscr{P}(\mathcal{U}))$, construct an instance of Knapsack (S, K):

- $S = \{1, 2, \ldots |\mathcal{U}|\}$
- $K = 2^{|\mathcal{U}|} - 1 = \sum_{i=0}^{|\mathcal{U}|-1}$

Interpret each set in $\mathcal{P}(S)$ as a bit vector.

Problem: Consider $S = \{1, 2, 3, 4\}$ and proposed cover $\{\{1, 3\}, \{1, 4\}, \{1\}\}$.

Independent Set problem: Given a graph, is there a set of vertices of size k with none adjacent to each other?

Reduction from 3Sat to Independent Set (LP pg 326–327.)

Suppose we have an instance of 3Sat, $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$. WOLOG, suppose each clause has exactly three literals. Construct an instance of Independent Set, (G, K) where $K = m$ and $G = (V, E)$ such that

- There is a vertex in V for each literal occurrence (or clause position) $c_{i,j}$.
- $(c_{i,j}, c_{x,y}) \in E$ if either
  - $i = x$ (they are positions in the same clause; this makes a triangle of vertices), or
  - the literals $c_{i,j}$ and $c_{x,y}$ are negations of each other.

Suppose an independent set of size K exists in G. It must include exactly one vertex in each triangle. Make a truth assignment that makes each literal in the set true.
Suppose a satisfying truth assignment exists. Then for each triangle, pick one vertex corresponding to a true literal.