Today’s Menu

1. Side Step: Constructing Witnesses if P=NP

2. Some Ways to Cope with NP-Completeness
Is the Restriction to Decision Problems Justified?

Almost all NP-complete problems are not really what we want:

- The answer $G \in \text{HAM}$ is insufficient. We rather want to find the cycle.
- The answer $(G, k) \in \text{CLIQUE}$ is insufficient. We rather want to find the largest clique.
- The answer $S \in \text{INTEGER PROGRAMMING}$ (the system $S$ of linear inequalities has an integer solution) is insufficient. We rather want to find the integer solution.
- The answer $\varphi \in \text{SAT}$ is insufficient. We rather want to find a satisfying assignment.
- ...

That is, we want to solve the corresponding function problems.
Is the Restriction to Decision Problems Justified?

What does $P = NP$ or $P \neq NP$ Tell Us about Solving Function Problems?

A function problem is at least as hard as its decision problem
$\Rightarrow$ if $P \neq NP$ then the function problems are hard as well, no chance.

But what if $P = NP$? Can even the corresponding function problems be efficiently solved in this case?
Example: SAT

Suppose $P = NP$, and let $M$ solve SAT in polynomial time.

How to find a satisfying assignment:

```
findAssignment(\varphi) \text{ where } \varphi \in SAT \text{ with variables } x_1, \ldots, x_n 
for i = 1, \ldots, n do 
  if $M(\varphi(\text{true}/x_i))$ then 
    ASS[i] \leftarrow \text{true} 
    \varphi \leftarrow \varphi(\text{true}/x_i) \quad \text{// replace } x_i \text{ by true} 
  else 
    ASS[i] \leftarrow \text{false} 
    \varphi \leftarrow \varphi(\text{false}/x_i) \quad \text{// replace } x_i \text{ by false} 
return ASS
```

Running time is roughly $n$ times the running time of $M$.
Finding Arbitrary Certificates if \( P = NP \)

Let \( R \in P \) be a polynomially bounded relation such that

\[
A = \{ x \mid \exists y \in \{0, 1\}^* \text{ such that } (x, y) \in R \}.
\]

\( \Rightarrow A' = \{ (x, y_0) \mid \exists y_1 \in \{0, 1\}^* \text{ such that } (x, y_0y_1) \in R \} \) is in \( NP \).

```plaintext
findCertificate(x) where x \in A

\[
\begin{align*}
y_0 & \leftarrow \epsilon \\
\text{while } (x, y_0) \not\in R \text{ do} & \\
\quad \text{if } (x, y_00) \in A' \text{ then} & \\
\qquad y_0 & \leftarrow y_00 \\
\quad \text{else} & \\
\qquad y_0 & \leftarrow y_01 \\
\text{return } y_0
\end{align*}
\]
```

Running time is again polynomial. [WHY?]
Let \( M \) be an \( n \times n \)-matrix of distances. Assume that the decision version \( \text{TSP}_D \) of the Travelling Salesman Problem is in \( P \).

**Step 1:** Find the length \( k \) of the shortest tour by binary search (similar to finding a certificate). Takes \( O(n) \) iterations since \( k \leq 2^n \).

**Step 2:**

```java
findTour(x)
    for i, j = 1, \ldots, n do
        M_{i,j} \leftarrow k + 1
        if \((M, k) \notin \text{TSP}_D\) then restore \( M_{i,j} \)
    return tour given by distances \( \leq k \) in \( M \)
```

Only \( n^2 \) iterations \( \Rightarrow \) running time is polynomial.
Some Ways to Cope with NP-Completeness

Ways to Cope with NP-Completeness

- **Brute Force**
  Try to find algorithms running in time $O(c^n)$ for a small $c$.

- **Randomization**
  Use algorithms that find the correct solution with high probability.

- **Approximation**
  Find a solution to an optimization problem not too far away from the optimum.

- **Fixed-parameter algorithms**
  Try to find algorithms that are efficient if certain parameters are small.
  (Next lecture)
An **independent set** in a graph $G$ is a set $V'$ of vertices such that none of them are connected by an edge.

**INDEPENDENT SET**

**Input:** A graph $G = (V, E)$  
**Output:** $\max\{|V'| \mid V' \subseteq V \text{ and } (u, v) \notin E \text{ for all } u, v \in V'\}$.

The straightforward solution tries all subsets of the set of vertices.  
$\Rightarrow$ time $O(2^n)$. 

---

*Frank Drewes (Umeå University)*  
Efficient Algorithms and Problem Complexity  
Lecture 12  
9 / 17
Some Ways to Cope with NP-Completeness

Example: Brute Force Algorithm for INDEPENDENT SET

Notation: $N(v) = \text{set of neighbors of } v \text{ in a graph}$

```
largestIndependentSet(G) \text{ where } G = (V, E)
if $E = \emptyset$ then
    return $|V|$ 
else
    Let $v \in V$ with $N(v) \neq \emptyset$
    $k \leftarrow largestIndependentSet(G - \{v\})$ // deselect $v$
    $l \leftarrow largestIndependentSet(G - (\{v\} \cup N(v)))$ // select $v$
    return $\max(k, l + 1)$
```

Recurrence: $T(0) = T(1) = 1 \text{ and } T(n + 2) = T(n + 1) + T(n) + O(n^2)$
⇒ analysis of the recurrence yields $T(n) \in O(1.62^n)$.

Best known: $O(1.211^n)$ (Robson’s algorithm).
Randomized Algorithms - the Class RP

Consider a polynomial-time RAM $M$ that has an additional coin flipping operation (fair and independent coin tosses).

$\Rightarrow M$ accepts an input $x$ with a certain probability $\Pr[M(x) = 1]$.

$M$ is a Monte Carlo algorithm for a decision problem $A$ if, for all $x$,

- if $x \in A$ then $\Pr[M(x) = 1] \geq 1/2$ and
- if $x \notin A$ then $\Pr[M(x) = 1] = 0$.

Yes is always correct, but no may be wrong with probability $1/2$.

Observation: The probability of wrong answers can be reduced to $2^{-k}$ by $k$ repetitions. E.g., $k = 100$ yields a failure probability smaller than $10^{-30}$.

Other randomized complexity classes: ZPP, PP, BPP.
Approximation Algorithms

We may want to approximate an optimization problem (here we consider the minimization case).

Minimization problems formalized

A minimization problem consists of a set \( I \) of instances, a set \( C(x) \) of solution candidates for every \( x \in I \), and a measure \( m: C \to \mathbb{R}_+ \) (where \( C = \bigcup_{x \in I} C(x) \)).

For \( x \in I \), the goal is to compute \( y \in C(x) \) such that \( m(y) = \text{opt}(x) = \min\{m(y') \mid y' \in C(x)\} \).

Suppose \( M \) computes some \( y \in C(x) \) for every \( x \in I \), but not necessarily one with \( m(y) = \text{opt}(x) \). The least upper bound of all

\[
m(M(x))/\text{opt}(x)
\]

is the performance ratio of \( M \).
Example: **BIN PACKING**

We want to store \( n \) items of size \( s_1, \ldots, s_n \leq 1 \) in as few as possible “bins” of size 1.

**BIN PACKING**

| Instance: | A sequence of positive item sizes \( s_1, \ldots, s_n \leq 1 \) |
| Candidate: | Placement of \( s_1, \ldots, s_n \) in \( m \) “bins” of size 1 each. |
| Measure: | The number \( m \) of bins used. |

The decision version (where \( m \) is given) is **BIN PACKING** is NP-complete.
An Approximation Algorithm for BIN PACKING

Johnsson’s algorithm: Fill the bins one by one with $s_1, \ldots, s_n$.

```
nextFit(s_1, \ldots, s_n)
allocate array store[1, \ldots, n]  // item i is stored in bin store[i]
bin \leftarrow 1; fill \leftarrow 0
for i = 1, \ldots, n do
    if fill + s_i > 1 then bin \leftarrow bin + 1
    store[i] \leftarrow bin
    fill \leftarrow fill + s_i
return store
```

- Running time is obviously linear.
- Performance ratio is 2: at most twice as many bins as necessary are used (next slide).
Performance Ratio of *nextFit*

```plaintext
nextFit(s_1, \ldots, s_n)
allocate array store[1, \ldots, n]  // item i is stored in bin store[i]
    bin ← 1; fill ← 0
    for i = 1, \ldots, n do
        if fill + s_i > 1 then bin ← bin + 1
        store[i] ← bin
        fill ← fill + s_i
    return store
```

Performance ratio: Suppose *nextFit* uses *b* bins.

The sum of the sizes of items in bins *i* and *i* + 1 is greater than 1
⇒ the total contents of all bins is at least *b/2*
⇒ at least \(\lceil b/2 \rceil\) bins are needed in every solution
⇒ in particular, the optimal solution uses at least \(\lceil b/2 \rceil\) bins.
Remarks Regarding \textit{nextFit}

- The performance ratio is indeed 2, i.e., for some instances, \textit{nextFit} uses (almost) twice as many bins as is optimal. [Can you find one?]
- It is an online algorithm: items are processed as they arrive.
- It is a 1-bounded-space algorithm: at most one bin is open at a time.

These are very useful properties. Think, e.g., of

- containers to be filled with delivered items, or
- data items to be combined into fixed-size packets and sent over communication channels.

If we precede \textit{nextFit} by sorting $s_1, \ldots, s_n$ decreasingly (offline!) the asymptotic performance ratio for large $n$ becomes $11/9$. 
Please read Sections 11.1–11.3 in the textbook.