These slides contain some extra clarification added post-lecture

NP-completeness recap

- NP is the class of problems we can check the answer to in polynomial time
- A problem \( A \) is NP-complete if
  - \( A \) is in NP
  - Every \( B \) in NP has a polynomial time reduction to \( A \) \((A \text{ is } NP\text{-hard})\)

If we could solve \( A \) quickly we could solve every problem in NP quickly

They are the ‘hardest’ problems in NP

we’re fairly certain you can’t solve them in polynomial time \((P \neq NP)\)

So if a problem is NP-complete, we give up right?

The Max-Sat problem

- Let \( \phi(x_1, x_2, \ldots, x_n) \) be an \( n \) variable, \( c \) clause boolean formula e.g.
  \[
  \phi(x_1, x_2, x_3, x_4) = (x_1 \lor x_2) \land (x_1 \lor x_3 \lor \neg x_4) \land (\neg x_1 \lor x_4 \lor x_2).
  \]
- The Max Sat (MaxSAT) problem is find the maximum number of clauses, \( M(\phi) \), which can be satisfied at the same time
- For example, \( x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 0 \) gives \( \phi(0,0,1,0) = (0 \lor 0) \land (0 \lor 1 \lor 1) \land (1 \lor 0 \lor 0) \). (two clauses satisfied)

The Max-Sat problem

- Let \( \phi(x_1, x_2, \ldots, x_n) \) be an \( n \) variable, \( c \) clause boolean formula e.g.
  \[
  \phi(x_1, x_2, x_3, x_4) = (x_1 \lor x_2) \land (x_1 \lor x_3 \lor \neg x_4) \land (\neg x_1 \lor x_4 \lor x_2).
  \]
- The Max Sat (MaxSAT) problem is find the maximum number of clauses, \( M(\phi) \), which can be satisfied at the same time
- Alternatively, \( x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 1 \) gives \( \phi(0,0,1,0) = (1 \lor 0) \land (1 \lor 1 \lor 1) \land (0 \lor 1 \lor 0) \). (three clauses satisfied)

Observe that \( M(\phi) = c \) iff \( \phi \) is satisfiable,

so Max-Sat is NP-hard

(and the decision version, ‘is \( M(\phi) \geq k \)’ is NP-complete)

The Max-Sat problem

- Let \( \phi(x_1, x_2, \ldots, x_n) \) be an \( n \) variable, \( c \) clause boolean formula e.g.
  \[
  \phi(x_1, x_2, x_3, x_4) = (x_1 \lor x_2) \land (x_1 \lor x_3 \lor \neg x_4) \land (\neg x_1 \lor x_4 \lor x_2).
  \]
- Find the maximum number of simultaneously satisfiable clauses, \( M(\phi) \)
- Can we approximate \( M(\phi) \)?
  - Let \( x_1 = x_2 = \ldots = x_n = 1 \) and count the satisfied clauses, \( m_1 \)
  - Let \( x_1 = x_2 = \ldots = x_n = 0 \) and count the satisfied clauses, \( m_0 \)
  - Output \( M'(\phi) = \max\{m_1, m_0\} \)

Claim \( M'(\phi) \geq M(\phi)/2 \)

- If \( m_0 \geq c/2 \), we are done (as \( c \geq M(\phi) \))
- Assume \( m_0 < c/2 \),
- Any clause not satisfied with all \( x_i = 0 \) is satisfied by all \( x_i = 1 \)
  
  so \( m_1 \geq c - m_0 \geq c/2 \)

The Max-Sat problem

- Let \( \phi(x_1, x_2, \ldots, x_n) \) be an \( n \) variable, \( c \) clause boolean formula e.g.
  \[
  \phi(x_1, x_2, x_3, x_4) = (x_1 \lor x_2) \land (x_1 \lor x_3 \lor \neg x_4) \land (\neg x_1 \lor x_4 \lor x_2).
  \]
- Find the maximum number of simultaneously satisfiable clauses, \( M(\phi) \)
- Can we approximate \( M(\phi) \)?
  - Let \( x_1 = x_2 = \ldots = x_n = 1 \) and count the satisfied clauses, \( m_1 \)
  - Let \( x_1 = x_2 = \ldots = x_n = 0 \) and count the satisfied clauses, \( m_0 \)
  - Output \( M'(\phi) = \max\{m_1, m_0\} \)

Claim \( M'(\phi) \geq M(\phi)/2 \)

So we can always find a solution which is at least \( 1/2 \) the optimal solution
even though Max-Sat is NP-hard
Approximation Algorithms

An algorithm $A$ is an $\alpha$-approximation for problem $P$ if,
- $A$ runs in polynomial time
- $A$ always outputs a solution with value $s$
  within an $\alpha$ factor of $\text{Opt}$
- Here $P$ is an optimisation problem with optimal solution of value $\text{Opt}$
- If $P$ is a maximisation problem (like MAXSAT), $\frac{\text{Opt}}{\alpha} \leq s \leq \text{Opt}$
- If $P$ is a minimisation problem, $\text{Opt} \leq s \leq \alpha \cdot \text{Opt}$
- We have seen a 2-approximation for MAXSAT
  The solution was between $\text{Opt}/2$ and $\text{Opt}$
- Here $\alpha$ is a constant but it could depend on $n$ (the input size)

Bin packing

$|\text{Bin}| = 1$ and there are an unlimited number of bins...

$0 < |\text{Item}| \leq 1$

$I$ is the sum of all item sizes

The Bin packing problem is known to be $\text{NP}$-hard
but we can approximate

If item $i$ fits into bin $j$: pack it, $i++$; else $j++$;
If item $i$ fits into bin $j$: pack it, $i++$; else $j++$.

Next fit runs in $O(n)$ time but how good is it?

Let $\text{fill}(i)$ be the sum of item sizes in bin $i$.

Observe that $\text{fill}(2i - 1) + \text{fill}(2i) > 1$ (for $1 \leq 2i \leq b$),

so $\lceil b/2 \rceil < \sum_{1 \leq 2i \leq b} \text{fill}(2i - 1) + \text{fill}(2i) \leq I \leq \text{Opt}$.

So Next fit is an 2-approximation for bin packing, which runs in linear time.

can we do better?

First fit decreasing (FFD)

Step 1: Sort the items into non-increasing order
First fit decreasing (FFD)

Step 2: Put each item in the first (left-most) bin it fits in

| 1 | 7/8 | 4/8 | 4/8 | 3/8 | 2/8 | 2/8 |

this is important for the proof

FFD runs in $O(n^2)$ time but how good is it?

• Consider bin $j = \left\lceil \frac{2}{b} \right\rceil$ (FFD uses $b$ bins on this instance)

the $j$-th bin from the left

Case 1: Bin $j$ contains an item of size $> 1/2$

Every bin $j' < j$ contains an item of size $> 1/2$

because we packed big things first and each thing was packed in the lowest numbered bin
First fit decreasing (FFD)

- Consider bin $j = \left\lceil \frac{j}{2} \right\rceil$ (FFD uses $b$ bins on this instance)
- Case 1: Bin $j$ contains an item of size $> 1/2$
  - Every bin $j' < j$ contains an item of size $> 1/2$
  - Each of these items has to have its own bin (even in Opt)
  - So $\text{Opt}$ uses at least $\frac{b}{2}$ bins
    - or... $b \leq 3\text{Opt}$

First fit decreasing (FFD)

- Consider bin $j = \left\lceil \frac{j}{2} \right\rceil$ (FFD uses $b$ bins on this instance)
- Case 2: Bin $j$ contains only items of size $\leq 1/2$
  - So bins $j, j+1, \ldots, b-1$ each contain at least two items
  - This gives a total of $2(b-j)$ items, none of which fits into bins $1, 2, \ldots, j-1$
    - otherwise we would have packed them there

First fit decreasing (FFD)

- Consider bin $j = \left\lceil \frac{j}{2} \right\rceil$ (FFD uses $b$ bins on this instance)
- Case 2: Bin $j$ contains only items of size $\leq 1/2$
  - So bins $j, j+1, \ldots, b-1$ each contain at least two items
  - This gives a total of $2(b-j) + 1$ items, none of which fits into bins $1, 2, \ldots, j-1$
    - $I > \min\{j-1, 2(b-j) + 1\} \geq \lceil 2b/3 \rceil - 1$
  
First fit decreasing (FFD)

- Consider bin $j = \left\lceil \frac{j}{2} \right\rceil$ (FFD uses $b$ bins on this instance)
- Case 2: Bin $j$ contains only items of size $\leq 1/2$
  - So bins $j, j+1, \ldots, b-1$ each contain at least two items
  - This gives a total of $2(b-j) + 1$ items, none of which fits into bins $1, 2, \ldots, j-1$
    - $I > \min\{j-1, 2(b-j) + 1\} \geq \lceil 2b/3 \rceil - 1$
    - (by plugging in $j = \lceil 2b/3 \rceil$)

First fit decreasing (FFD)

- Consider bin $j = \left\lceil \frac{j}{2} \right\rceil$ (FFD uses $b$ bins on this instance)
- Case 2: Bin $j$ contains only items of size $\leq 1/2$
  - So bins $j, j+1, \ldots, b-1$ each contain at least two items
  - This gives a total of $2(b-j) + 1$ items, none of which fits into bins $1, 2, \ldots, j-1$
    - $I > \min\{j-1, 2(b-j) + 1\} \geq \lceil 2b/3 \rceil - 1$
    - As $\lceil 2b/3 \rceil - 1 < I$ and $I \leq \text{Opt}$
      - we have that $\lceil 2b/3 \rceil - 1 < \text{Opt}$
      - ...but both sides are integers...
      - so $\lceil 2b/3 \rceil \leq \text{Opt}$

Finally...

- ...$2b/3 \leq \lceil 2b/3 \rceil \leq \text{Opt}$
- or $b \leq (3/2)\text{Opt}$
First fit decreasing (FFD)

1

\[ \begin{array}{c|c|c|c}
7/8 & 4/8 & 2/8 & \cdots \\
4/8 & 3/8 & & \\
\end{array} \]

Consider \( j = \left\lceil \frac{2b}{3} \right\rceil \) (FFD uses \( b \) bins on this instance)

Case 1: Bin \( j \) contains an item of size \( > \frac{1}{2} \)

Case 2: Bin \( j \) contains only items of size \( \leq \frac{1}{2} \)

in both cases… \( b \leq \frac{3\text{Opt}}{2} \)

So FFD is a \( 3/2 \)-approximation for bin packing

Conclusions

We saw approximations for several \( \text{NP} \)-hard problems,

An \( O(n) \) time \( 2 \)-approximation for Bin packing
An \( O(n^2) \) time \( 3/2 \)-approximation for Bin packing
A \( 2 \)-approximation for MaxSAT