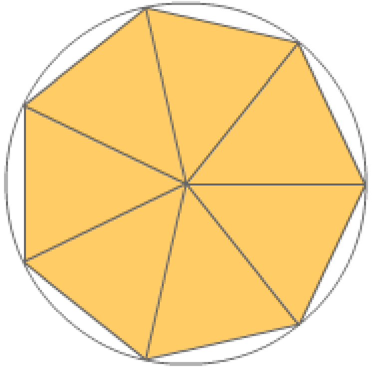
Coping with NP-Completeness



Theory of Computation

CISC 603, Spring 2020, Daqing Yun

3-SAT

PROBLEM 13.1. **3-SAT Problem.**
Input: given a boolean formula in format of

$$\psi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge \cdots \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3),$$

where every clause contains at most 3 literals (boolean variables).
Question: is there an assignment to x_1, x_2, x_3 that makes ψ is true?

brute-force? $O(2^n)$ Can we do better?

Observations

PROBLEM 13.1. **3-SAT Problem.**
Input: given a boolean formula in format of

$$\psi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge \cdots \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3),$$

where every clause contains at most 3 literals (boolean variables).
Question: is there an assignment to x_1, x_2, x_3 that makes ψ is true?

- Do we check certain clauses multiple times?
- Do all clauses contain three literal variables?
- We have tried *divide and conquer*, how about *decrease and conquer*?

Let $t(x_i)$ be the evaluated value of literal x_i or $\overline{x_i}$ $(i \in \{1, 2, 3\})$.
let $a = t(x_1)$, $b = t(x_2)$, $c = t(x_3)$

Let $t(x_i)$ be the evaluated value of literal x_i or $\overline{x_i}$ $(i \in \{1, 2, 3\})$.
let $a = t(x_1)$, $b = t(x_2)$, $c = t(x_3)$

Algorithm 1 3-SAT(ψ)

if ψ is empty **then**
    return True;
if ψ has a clause contains one literal a **then**
    $\psi = \psi[a \text{ is True}]$
    return 3-SAT(ψ)
if ψ has a clause contains two literals a and b **then**
    $\psi_1 = \psi[a \text{ is False}, b \text{ is True}]$
    $\psi_2 = \psi[a \text{ is True}]$
    return 3-SAT(ψ_1) $\cup$ 3-SAT(ψ_2)
if ψ has a clause contains three literals a, b, and c **then**
    $\psi_1 = \psi[a \text{ is False}, b \text{ is False}, c \text{ is True}]$
    $\psi_2 = \psi[a \text{ is False}, b \text{ is True}]$
    $\psi_3 = \psi[a \text{ is True}]$
    return 3-SAT(ψ_1) $\cup$ 3-SAT(ψ_2) $\cup$ 3-SAT(ψ_3)

Time Complexity

- Did we really improve it?

The running time complexity of Algorithm 1 is

$$T(n) = T(n-1) + T(n-2) + T(n-3).$$

by solving equation $x^n = x^{n-1} + x^{n-2} + x^{n-3}$, we get $x \approx 1.84$, thus

$$T(n) = \mathcal{O}(1.84^n).$$

Largest Independent Set

DEFINITION 13.2. An *independent set* S of a graph $G(V, E)$ is a subset of the vertices in G such that there is no edge between any two vertices in S.

DEFINITION 13.3. **Independent Set Problem**
Instance: a graph $G(V, E)$, integer $k \leq |V|$.
Question: does G contain an independent set of size at least k?

PROBLEM 13.4. **Largest Independent Set Problem**
Input: a graph $G(V, E)$.
Question: what is the size of the largest independent set of $G(V, E)$?

Brute-Force

Using a brute-force approach, as for each vertex $v \in G$, there are two possibilities, i.e., either v is in the largest independent set or not.

Let us use a V-bit binary number x to represent the solution, each bit $x_i (i = 1, 2, \cdots, |V|)$ of x indicates if a vertex v_i is in the largest independent set or not, i.e., either 1 (in the set) or 0 (not in the set).

$O(2^n)$

I mean "so far" in this class

The best we can do so far is to try to reduce the base of the exponential running time complexity, i.e., reduce x in $\mathcal{O}(x^n)$.

It seems that there is no short cut but repeatedly picking and checking

What if we pick a vertex *v* arbitrarily?

If we pick v arbitrarily without considering its degree at Line 3, the running time of complexity of Algorithm 2 is $T(n) = 2T(n-1)$, i.e., $T(n)$ is still $\mathcal{O}(2^n)$.

What if we pick a vertex *v* with non-zero degree?

Algorithm 2 LISBruteForce($G(V, E)$)

1: **if** E is $\emptyset$ **then**
2:     **return** $|V|$
3: Pick $v \in V$ such that $N(v)$ is not $\emptyset$
4: $G_1 = G - \{v\}$
5: $G_2 = G - \{v\} - \{N(v)\}$
6: $k_1 = \text{LIS}(G_1)$
7: $k_2 = \text{LIS}(G_2)$
8: **return** $\max\{k_1, k_2 + 1\}$

let $N(v)$ denote the set of neighbouring vertices of vertex v

Time Complexity

$$T(n) = T(n-1) + T(n-2).$$

$$T(n) = \mathcal{O}(x^n) \qquad x^n = x^{n-1} + x^{n-2}, \quad T(n) = \mathcal{O}(1.618^n).$$

Algorithm 2 LISBruteForce($G(V, E)$)

```
if E is ∅ then
    return |V|
Pick v ∈ V such that N(v) is not ∅
G_1 = G − {v}
G_2 = G − {v} − {N(v)}
k_1 = LIS(G_1)
k_2 = LIS(G_2)
return max{k_1, k_2 + 1}
```

Approximation?

- Very hard for general cases unless P is equal to NP
- Focus on special cases, try a greedy approach

Note: this algorithm runs in polynomial time, but may not give us optimal solution

Algorithm 3 LISGreedy$(G(V, E))$

$S \leftarrow \emptyset$
while G is not empty **do**
  Let v be a node with minimal degree in G;
  $S \leftarrow S \cup \{v\}$;
  Remove v and its neighbors from G
return S

Approximation?

- How bad can it be?
- When picking *v*, how many nodes can be removed at most thus can not be in the IS? Call this number Δ
- We have $|V - S| \leq \Delta \cdot |S|$ and $S + |V - S| = n$, thus

$$\Delta \cdot |S| + |S| \geq n \Rightarrow |S| \cdot (\Delta + 1) \geq n, \text{ i.e., } |S| \geq \frac{n}{\Delta + 1}$$

Algorithm 3 LISGreedy$(G(V, E))$

```
S ← ∅
while G is not empty do
    Let v be a node with minimal degree in G;
    S ← S ∪ {v};
    Remove v and its neighbors from G
return S
```

a $\frac{1}{\Delta+1}$-approximation algorithm

Bin-Packing

PROBLEM 13.5. **Bin-Packing Problem**
Input: given n items with weights $a_1, a_2, \ldots, a_n$ $(1 > a_1 \geq a_2 \geq \ldots \geq a_n > 0)$, and m bins that can hold any subset of the items with total weight up to 1;
Question: find out the minimal number of bins needed to pack all n items.

Let us also try to design greedy algorithms

Bin-Packing

PROBLEM 13.5. **Bin-Packing Problem**
Input: given n items with weights $a_1, a_2, \ldots, a_n$ $(1 > a_1 \geq a_2 \geq \ldots \geq a_n > 0)$, and m bins that can hold any subset of the items with total weight up to 1;
Question: find out the minimal number of bins needed to pack all n items.

We open up bins one by one and consider items in their index-increasing order.

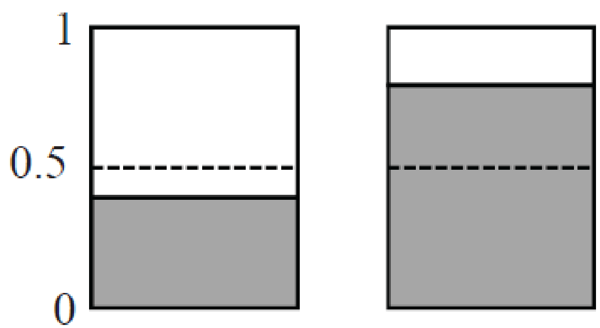
We open bin 1 and put item 1 into it, consider bin 1 as the current bin and check next item: if the item fits the current opened bin, put it into the current bin;

otherwise, close the current bin and put the item into a newly-opened bin and consider this new bin as the current bin. Take next item and repeat checking.

Linear time, $O(n)$

Approximation?

Problem 13.5. **Bin-Packing Problem**
Input: given n items with weights $a_1, a_2, \ldots, a_n$ $(1 > a_1 \geq a_2 \geq \ldots \geq a_n > 0)$, and m bins that can hold any subset of the items with total weight up to 1;
Question: find out the minimal number of bins needed to pack all n items.



$$\boxed{\mathrm{OPT} \leq \mathrm{NF} \leq 2 \cdot \sum a_i \leq 2 \cdot \mathrm{OPT}\,.}$$

Notice that it is impossible that any two consecutive bins are both filled with items of total weight ≤ 0.5

Any two consecutive bins are filled with items whose total weight is ≥ 1

To fill in all items with a total weight $\sum a_i$, we need $\mathrm{NF} \leq 2 \cdot \sum a_i$ bins.

$\sum a_i$ is the lower bound in which case no space in any bin is left "wasted"

LIS

- Formulate LIS as an LP
- Using rounding to get a feasible solution

LIS

- Formulate LIS as an LP
- Using rounding to get a feasible solution

Define a variable x_v associated with a node $v \in V$ as,

$$x_v = \begin{cases} 1, & \text{if } v \text{ is in the LIS} \\ 0, & \text{otherwise,} \end{cases} \tag{13.3}$$

thus the corresponding transformed LP is given in Problem 13.6.

PROBLEM 13.6. **LIS Problem as an LP**.

$$\max \sum_v x_v \,, \tag{13.4}$$

subject to

$$x_u + x_v \leq 1, \text{ for all } (u, v) \in E \,, \tag{13.5}$$

$$0 \leq x_v \leq 1, \text{ for all } v \in V \,. \tag{13.6}$$

Minimal Vertex Cover

A *vertex cover* of a graph $G(V, E)$ is a subset U of V such that every edge of G has at least one endpoint in U [4].

PROBLEM 13.7. **Minimal Vertex Cover Problem**
Input: a graph $G(V, E)$, an integer k.
Question: is there a vertex cover U of G including at most k vertices?

Algorithm 4 VCBruteForce($G(V, E), k$)

1: **if** $k == 0$ **then**
2:     **return** $(|E| == 0)$
3: Pick an edge $(u, v) \in E$
4: $G_1 = (V - \{u\},\ E - \{(u, w) | w \in V\})$ // u is in VC
5: $G_2 = (V - \{v\},\ E - \{(v, w) | w \in V\})$ // v is in VC
6: **return** VC$(G_1, k-1) \cup$ VC$(G_2, k-1)$

$$T(n, k) = 2T(n-1, k-1) \leq 2^2 T(n-2, k-2) \ldots$$

$$= 2^{k-1} T(n-(k-1), 1) = 2^{k-1} \cdot (n-(k-1)) = \mathcal{O}(2^k \cdot n)$$

Minimal Vertex Cover

- Eliminating the *n* from $\mathcal{O}(2^k \cdot n)$ using preprocessing

In observation of that if u's degree is larger than k, i.e., $degree(u) > k$, then u must be in the VC if there is one exists, otherwise, a VC does not exist.

Suppose there are m vertices whose degrees are larger than k.

Algorithm 5 VCFaster$(G(V, E), k)$

Find all m vertices with degrees larger than k
if $m > k$ **then**
    return False
Remove these m vertices and corresponding edges from G
Remove vertices with 0 degree, i.e., the isolated vertices
Denote the updated graph as G'
if $|V'| > 2k(k-m)$ **then**
    return False
return VC$(G', k-m)$

$$\boxed{T(n) = \mathcal{O}(2^k \cdot n) = \mathcal{O}(2^k \cdot 2k(k-m)) = \mathcal{O}(2^k \cdot k^2).}$$

MVC

- MVC as an ILP

MVC problem could be expressed as an ILP as follows, where $x_i \in \{0,1\}$ with $x = 1$ indicates $v_i \in U$ and 0 otherwise:

$$\min \sum_{i=1}^{|V|} x_i,$$

subject to

$$x_i + x_j \geq 1, \text{ for each edge } e(i,j),\ x_i \in \{0,1\}.$$

MVC

- Get a feasible solution via rounding

Now we change the restrictions to be

$$x_i + x_j \geq 1, \text{ for each edge } e(i,j),\ 0 \leq x_i \leq 1,$$

and get the corresponding LP.

Solving this LP in polynomial, we get its optimal solution, denoted as

$$\mathbf{x} = (x_1, x_2, \ldots, x_{|V|}),\ 0 \leq x_i \leq 1,\ \text{for } i = 1, 2, \ldots, |V|. \tag{13.7}$$

We use a rounding method: if $x_i \geq 0.5$, then x_i is rounded up to 1.0, otherwise x_i is rounded down to 0.0, to get a feasible solution to MVC, denoted as

$$\mathbf{x}^* = (x_1^*, x_2^*, \ldots, x_{|V|}^*),\ x_i^* \in \{0, 1\},\ \text{for } i = 1, 2, \ldots, |V|. \tag{13.8}$$

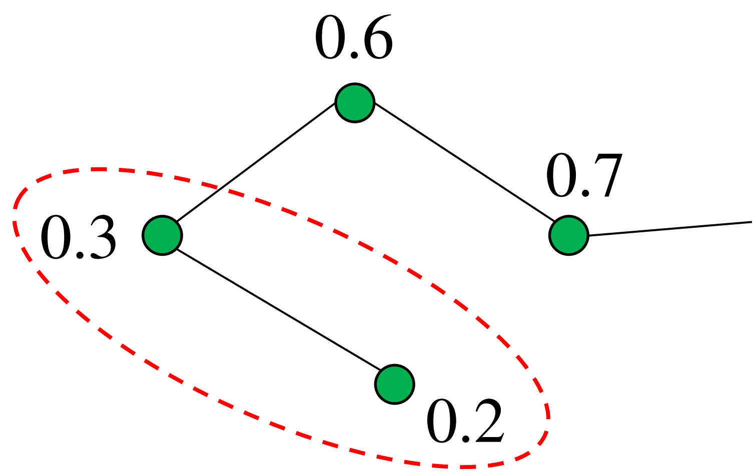
Note that $\mathbf{x}^*$ must produce a vertex cover (i.e., a feasible solution) due to the constraint $x_i + x_j \geq 1$ that indicates at least one of x_i or x_j has to be ≥ 0.5.

Approximation?

$\mathrm{OPT}_{\mathrm{ILP}} \le \sum_{i=1}^{|V|} x_i^*$

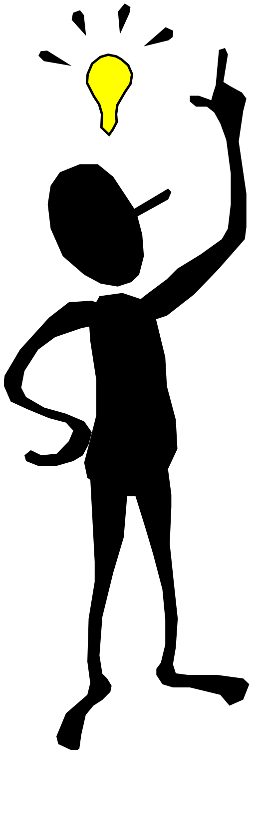
$\mathrm{OPT}_{\mathrm{LP}} \le \mathrm{OPT}_{\mathrm{ILP}}$

$x_i^* \le 2x_i$



$$\mathrm{OPT}_{\mathrm{ILP}} \le \sum_{i=1}^{|V|} x_i^* \le$$

$$\sum_{i=1}^{|V|} 2x_i = 2\sum_{i=1}^{|V|} x_i = 2 \cdot \mathrm{OPT}_{\mathrm{LP}} \le 2 \cdot \mathrm{OPT}_{\mathrm{ILP}}$$



Thanks ! ☺

Questions ?