

10.1 Approximation Algorithms

- Let us examine a problem, where we are given
 - A ground set U with m elements
 - A collection of subsets of the ground set $S = \{S_1, \dots, S_n\}$ s.t. it is a cover of U : $US = U$
- The aim is to find a subcover $S' \subseteq S$,
 $US' = U$,
 containing as few subsets as possible
- This problem is known as the **Minimum Set Cover** (minSC)
- One of the oldest and most studied combinatorial optimization problems



- The corresponding decision problem
 - Given: a ground set U , cover S and a natural number k
 - Question: Does U have a subcover $S' \subseteq S$ s.t. $|S'| \leq k$?

Theorem *The decision version of minimum set cover problem is NP-complete.*

Proof. Obviously $\text{minSC} \in \text{NP}$: Let us guess from the given cover S a subcover S' containing k subsets and verify deterministically in polynomial time that we really have a subcover.



Polynomial time reduction $VC \leq_m^p \text{minSC}$ is easy to give. Let $\langle G, k \rangle$ be an instance of the vertex cover in which $G = (V, E)$. We choose the mapping f :

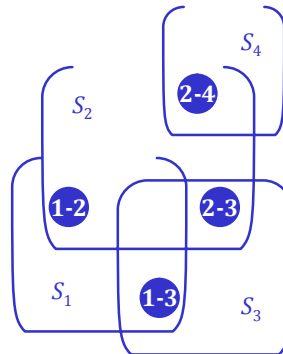
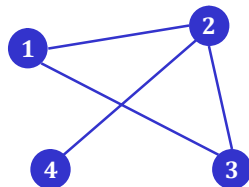
$$f(\langle (V, E), k \rangle) = \langle E, V_E, k \rangle,$$

where V_E is the collection of edges connected to the nodes of G .

In other words, for each $v \in V$ has a corresponding set

$$\{ e \in E \mid e = (v, w) \}.$$

Clearly f is computable in polynomial time and is a reduction. \square





- Hence, minSC is an intractable problem – we do not know of a polynomial time algorithm for solving it
- Therefore, we attempt to find a polynomial time algorithm
 - that does not necessarily give the best possible (optimal) solution, but
 - can be shown always to be at most a function of the input length worse than the optimal solution
- Such an algorithm is called an **approximation algorithm**
- Let us denote by
 - Opt the cost of the solution given by an optimal algorithm and
 - App that of the solution given by an approximation algorithm



- Since minSC is a minimization problem, $App/Opt \geq 1$
- The closer to 1 this ratio is, the better the solution produced approximates the optimal solution
- From an approximation algorithm one requires that the fraction is bounded by a function of the length n of the input

$$\frac{App}{Opt} \leq \rho(n)$$

- $\rho(n)$ is the **approximation ratio** of the algorithm
 - The algorithm is called an $\rho(n)$ -approximation algorithm
 - At the best the approximation ratio does not depend at all on the length n of the input, but is constant



- Let us examine the following algorithm for **vertex cover**
- We will show that it is an 2-approximation algorithm for the problem

Input: An undirected graph $G = (V, E)$

Output: Vertex cover C

1. $C \leftarrow \emptyset$;

2. $E' \leftarrow E$;

3. **while** $E' \neq \emptyset$ **do**

a. Let (u, v) be any edge of the set E' ;

b. $C \leftarrow C \cup \{u, v\}$;

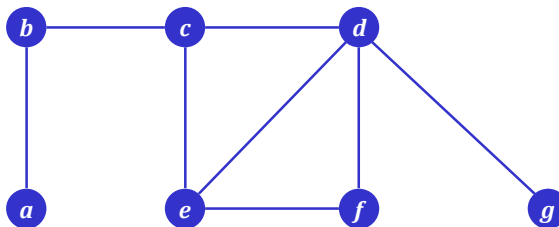
c. Remove from E' all edges connected to nodes u and v ;

4. **od**;

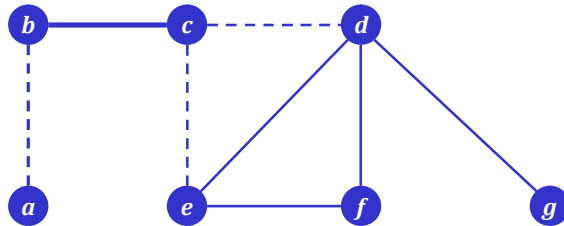
5. **return** C ;



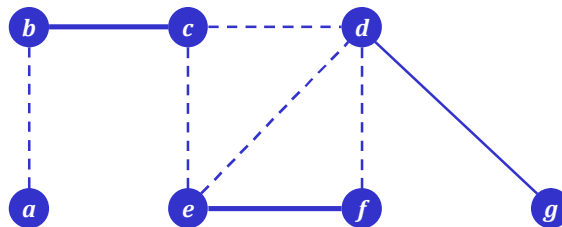
Selection of the first random edge: (b, c)



We remove other edges connected with nodes b and c

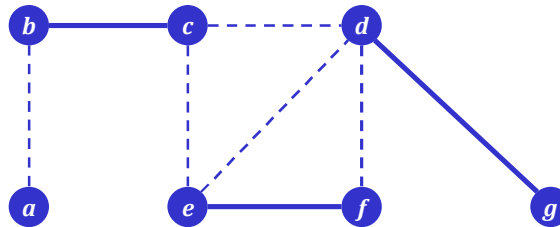


The next random choice : (e, f) and
Removal of other edges connected with its nodes



The only remaining choice (d, g)

We end up with a cover of 6 nodes,
while the optimal one has 3 nodes (e.g., b, d, e)



Theorem 10.1 *The above given algorithm is polynomial time 2-approximation algorithm for vertex cover.*

Proof. The time complexity of the algorithm, using adjacency list representation for the graph, is $O(V + E)$, and thus uses a polynomial time.

The set of nodes C returned by the algorithm obviously is a vertex cover for the edges of G , because nodes are inserted into C in the loop of row 3 until all edges have been covered.

Let A be the set of edges chosen by algorithm in row 3a. In order to cover the edges of A any vertex cover — in particular also the optimal vertex cover — has to contain at least one of the ends of each edge in A .





Because the end points of the edges in A are distinct by the design of the algorithm, $|A|$ is a lower bound for the size of any vertex cover.

In particular,

$$\text{Opt} \geq |A|.$$

The above algorithm always selects in row 3a an edge whose neither end point is yet in the set C . Hence,

$$\text{App} = |C| = 2|A|.$$

Combining the above equations yields

$$\text{App} = 2|A| \leq 2 \text{Opt},$$

and therefore

$$\text{App}/\text{Opt} \leq 2.$$

□



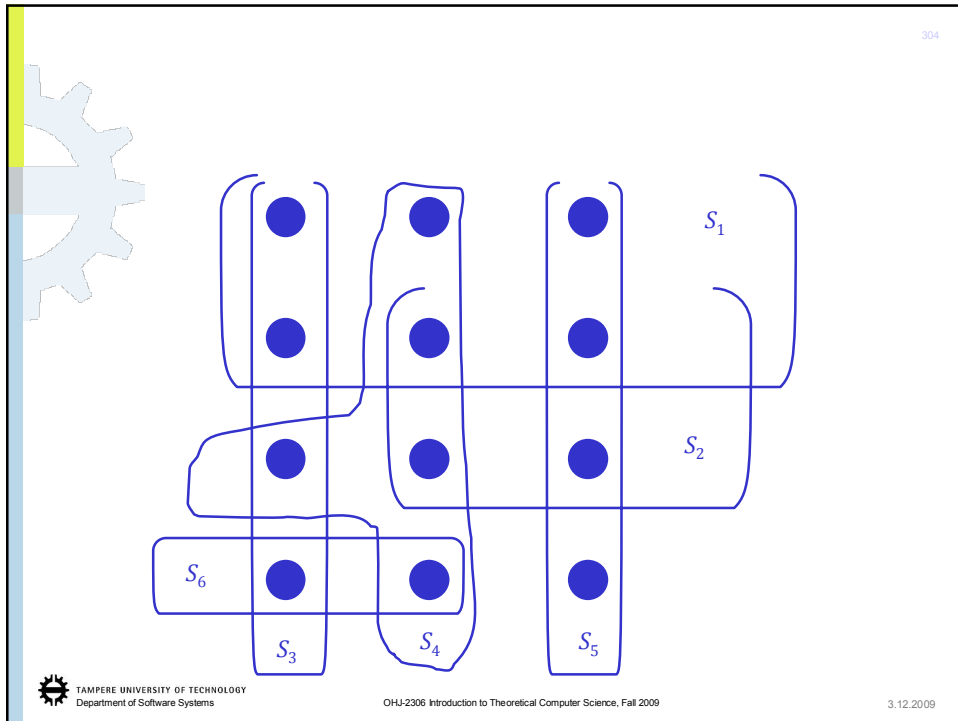
- Also set cover has a simple greedy approximation algorithm
- Neither this nor any other polynomial time deterministic algorithm can attain a constant approximation ratio

Input: Ground set U and its cover S

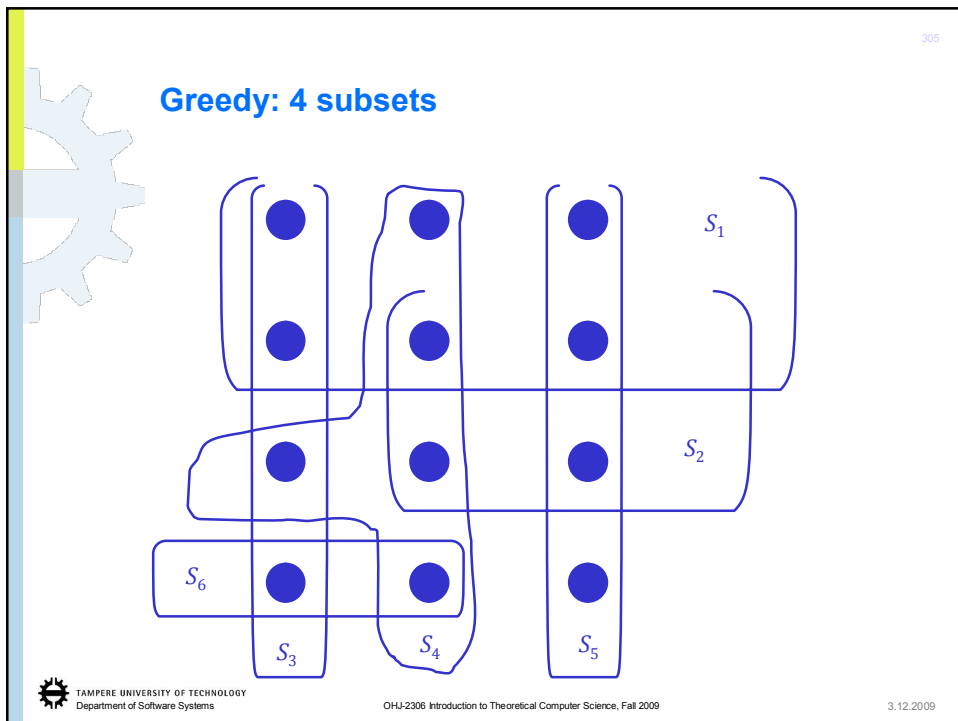
Output: Set cover C

1. $X \leftarrow U; C \leftarrow \emptyset;$
2. **while** $X \neq \emptyset$ **do**
 - a. select $S' \in S$ s.t. $|S' \cap X|$ is maximized;
 - b. $X \leftarrow X \setminus S';$
 - c. $C \leftarrow C \cup \{S'\};$
3. **od;**
4. **return** $C;$

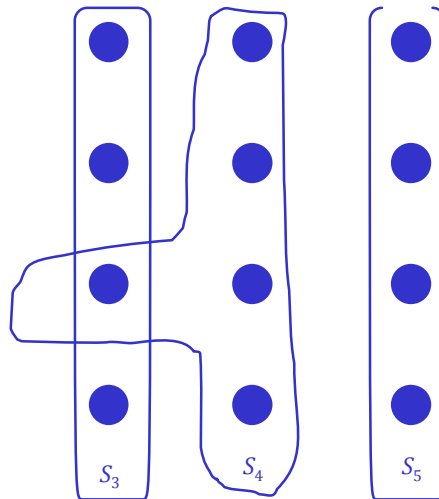




Greedy: 4 subsets



Optimal: 3 subsets



- The greedy algorithm can quite easy to implement to run in polynomial time in the length of the input $|U|$ and $|S|$
 - The loop in row 2 is executed at most $\min(|U|, |S|)$ times and the body of the loop itself can be implemented to require time $O(|U| \cdot |S|)$
 - Altogether the time requirement thus is $O(|U| \cdot |S| \min(|U|, |S|))$
 - It is also possible to give a linear time implementation for the greedy approximation algorithm for set cover
- The collection C returned by the algorithm is obviously a set cover, because the loop of row 2 is executed until there are no more elements to cover

- In order to relate the cost of the set cover returned by the greedy algorithm, we set cost 1 to each of the chosen sets
- Let S_i be the set selected by the greedy algorithm at round i
- We distribute the cost of S_i evenly among all those elements in it that now become covered for the first time
- Let c_u denote the cost assigned on element $u \in U$
- Each element gets assigned a cost only once, the first time it is covered by some set
- If u is first covered by the set S_i , the cost assigned to it is:

$$c_u = \frac{1}{|S_i \setminus (S_1 \cup S_2 \cup \dots \cup S_{i-1})|}$$



- Each set selected by the greedy algorithm is assigned cost 1 so that

$$\text{App} = |C| = \sum_{u \in U} c_u$$

- On the other hand, the cost of the optimal cover C^* is

$$\sum_{S' \in C^*} \sum_{u \in S'} c_u$$

- Because each $u \in U$ belongs to at least one $S' \in C^*$, we have

$$\sum_{S' \in C^*} \sum_{u \in S'} c_u \geq \sum_{u \in U} c_u$$

- Combining the above given yields

$$\text{App} \leq \sum_{S' \in C^*} \sum_{u \in S'} c_u$$



- Let $H(k)$ denote the k -th harmonic number

$$H(k) = \sum_{j=1}^k \frac{1}{j} = 1 + \frac{1}{2} + \dots + \frac{1}{k}$$

- We define $H(0) = 0$
- Next we show that for any $S' \in \mathcal{S}$ it holds

$$\sum_{u \in S'} c_u \leq H(|S'|)$$

- Then, by the previous inequality,

$$\begin{aligned} \text{App} &\leq \sum_{S' \in \mathcal{C}^*} H(|S'|) \\ &\leq |\mathcal{C}^*| \cdot H(\max\{|S'| : S' \in \mathcal{S}\}) \\ &= \text{Opt} \cdot H(\max\{|S'| : S' \in \mathcal{S}\}) \end{aligned}$$



Lemma For each $S' \in \mathcal{S}$ it holds

$$\sum_{u \in S'} c_u \leq H(|S'|)$$

Proof. Let $S' \in \mathcal{S}$ be arbitrary and $i = 1, 2, \dots, |C|$. Furthermore, let

$$n_i = |S' \setminus (S_1 \cup S_2 \cup \dots \cup S_i)|$$

be the number of those elements of S' that have not yet been covered when the greedy algorithm has chosen sets S_1, S_2, \dots, S_i to the set cover.

Let $n_0 = |S'|$.

Let k be the smallest index s.t. $n_k = 0$; i.e., every element of S' belongs to at least one of the sets S_1, S_2, \dots, S_k .

Then $n_{i-1} \geq n_i$ and $S_i, i = 1, 2, \dots, k$, covers $n_{i-1} - n_i$ elements for the first time.



Now

$$\sum_{u \in S'} c_u = \sum_{i=1}^k (n_{i-1} - n_i) \frac{1}{|S_i \setminus (S_1 \cup \dots \cup S_{i-1})|}.$$

Since S_i is chosen greedily, it must cover at least as many elements as the set S' (or otherwise S' should have been selected). Hence,

$$|S_i \setminus (S_1 \cup \dots \cup S_{i-1})| \geq |S' \setminus (S_1 \cup \dots \cup S_{i-1})| = n_{i-1}$$

Which further yields

$$\sum_{u \in S'} c_u \leq \sum_{i=1}^k (n_{i-1} - n_i) \frac{1}{n_{i-1}}.$$



$$\begin{aligned} \sum_{u \in S'} c_u &\leq \sum_{i=1}^k (n_{i-1} - n_i) \frac{1}{n_{i-1}} \\ &= \sum_{i=1}^k \sum_{j=n_i+1}^{n_{i-1}} \frac{1}{n_{i-1}} \\ &\leq \sum_{i=1}^k \sum_{j=n_i+1}^{n_{i-1}} \frac{1}{j}, \end{aligned}$$

because $j \leq n_{i-1}$. Moreover,

$$\begin{aligned} &= \sum_{i=1}^k \left(\sum_{j=1}^{n_{i-1}} \frac{1}{j} - \sum_{j=1}^{n_i} \frac{1}{j} \right) \\ &= \sum_{i=1}^k (H(n_{i-1}) - H(n_i)) \\ &= H(n_0) - H(n_k), \end{aligned}$$

since the other terms in the sum cancel each other out.





We have chosen $n_k = 0$ and defined $H(0) = 0$. Therefore, further

$$\begin{aligned} &= H(n_0) - H(0) \\ &= H(n_0) \\ &= H(|S'|) \end{aligned}$$

and we have proved the lemma. \square

- For the harmonic number $H(k)$ it holds $\ln k < H(k) \leq \ln k + 1$
- From the above results it follows:

Theorem *For the greedy algorithm of the set cover problem it holds that*

$$\frac{\text{App}}{\text{Opt}} \leq H(\max\{|S'| : S' \in \mathcal{S}\}) \leq \ln |U| + 1$$



- In some applications $\max\{|S'| : S' \in \mathcal{S}\}$ is a small constant
- Then the solution returned by the greedy algorithm is only a small constant away from the optimal one
- In particular, if subsets S' have an upper bound d for their size,
 $\text{App}/\text{Opt} \leq H(d)$
- E.g., when the nodes of the graph of vertex cover have maximum degree 3, then
 - the solution returned by the greedy set cover algorithm is at most $H(3) = 11/6 < 2$ times as large as the optimal cover





- Feige, 1996: no polynomial-time algorithm can approximate minSC within $(1-\epsilon) \ln m$, for any $\epsilon > 0$, unless $NP \subseteq DTIME(n^{\log \log n})$
- Hence, it is not possible to find an approximation algorithm for minSC that would be significantly better than the greedy one
- Slavík, 1996: A more exact upper bound for the approximation ratio of the greedy algorithm is

$$\ln m - \ln \ln m + \Theta(1)$$
- In fact this is also a lower bound for the approximation ratio of the greedy algorithm
- $\ln m - \ln \ln m + \Theta(1)$ is thus the asymptotically exact approximation ratio of the greedy algorithm



Course Recap

- ✓ All computational problems cannot be solved algorithmically
- ✓ A deterministic finite automaton (DFA) has a unique minimal automaton. The minimal automaton can be constructed in a straightforward manner
- ✓ Nondeterministic finite automata (NFA) do not recognize more languages than DFAs
- ✓ A language is regular
 - ⇔ it can be recognized with a finite automaton
 - ⇔ it can be described with a regular expression
 - ⇔ it can be generated with a right-linear grammar



- ✓ Strings of a regular language can be "pumped"
- ✓ There are sensible formal languages that are not regular
- ✓ Context-free languages are a proper superset of regular languages
- ✓ A language is context-free if and only if it can be recognized with a pushdown automaton (PDA)

- ✓ By the Church-Turing thesis any problem solvable on a computer can also be solved using a Turing machine (TM)
- ✓ Variants of Turing machines – including nondeterministic Turing machines – have equal recognition power to the standard single-tape machine



- ✓ The "efficiency" of different machines varies
- ✓ Languages generated by unrestricted grammars are equivalent to those recognized by Turing machines
- ✓ A total Turing machine (decider) halts on every input
- ✓ A formal language is Turing-recognizable (TR), if it can be recognized with a TM and Turing-decidable, if it has a decider

- ✓ A and B decidable $\Leftrightarrow \bar{A}$, $A \cup B$ and $A \cap B$ are decidable
- ✓ $A, B \in \text{TR} \Leftrightarrow (A \cup B), (A \cap B) \in \text{TR}$
- ✓ A decidable $\Leftrightarrow A, \bar{A} \in \text{TR}$
- ✓ $A \in \text{TR}$, not decidable $\Leftrightarrow \bar{A} \notin \text{TR}$





- ✓ $D = \{c \in \{0, 1\}^* \mid c \notin L(M_c)\} \notin \text{TR}$
- ✓ E.g., A_{DFA} , E_{DFA} , and EQ_{DFA} are decidable languages

- ✓ $U = \{\langle M, w \rangle \mid w \in L(M)\} \in \text{TR}$, not decidable
- ✓ $\bar{U} = \{\langle M, w \rangle \mid w \notin L(M)\} \notin \text{TR}$

- ✓ $H = \{\langle M, w \rangle \mid M(w) \downarrow\} \in \text{TR}$, not decidable
- ✓ $\bar{H} = \{\langle M, w \rangle \mid M(w) \uparrow\} \notin \text{TR}$

- ✓ Chomsky hierarchy:
 finite $\not\subseteq$ regular $\not\subseteq$ context-free $\not\subseteq$ context-sensitive $\not\subseteq$ languages
 generated by unrestricted grammars (= TR)



- ✓ $NE = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \neq \emptyset\} \in \text{TR}$, not decidable
- ✓ $REG = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is regular}\}$ is not decidable
- ✓ Rice's theorem: All nontrivial semantic properties of Turing machines are undecidable

- ✓ Linear bounded automaton (LRA) cannot use more work space than that already required by input
- ✓ A_{LRA} is decidable, while E_{LRA} is undecidable

- ✓ $A \subseteq \Sigma^*$ is *reducible* to B , $B \subseteq \Gamma^*$, denoted $A \leq_m B$,
 if there exists a computable function $f: \Sigma^* \rightarrow \Gamma^*$ s.t.

$$x \in A \Leftrightarrow f(x) \in B \quad \forall x \in \Sigma^*$$





✓ $A \subseteq \{0, 1\}^*$ is *TR-complete*, if

1. $A \in \text{TR}$ and
2. $B \leq_m A$ for all $B \in \text{TR}$

- ✓ Language U is TR-complete
- ✓ In time complexity analysis of Turing machines one examines the worst case with inputs on certain length
- ✓ Relating growth rates of functions: O, Θ, o, Ω
- ✓ The number of tapes does not have a significant impact on the efficiency of a Turing machine
- ✓ The efficiency difference of a deterministic and nondeterministic TM, on the other hand, is exponential



$$P = \bigcup_{k \geq 0} \text{DTIME}(n^k + k)$$

$$\text{EXPTIME} = \bigcup_{k \geq 0} \text{DTIME}(2^{n^k})$$

- ✓ P includes languages that can be decided in time that is polynomial in the length of the input
- ✓ E.g., finding a directed path in a graph, PATH, and deciding whether two numbers are relatively prime are examples of problems in P
- ✓ The corresponding nondeterministic composite classes are NP and $NEXPTIME$
- ✓ For instance, CLIQUE and SUBSET-SUM are problems in NP
- ✓ Problems belonging to P are solvable in practice

- ✓ $P \subseteq NP$. In addition NP contains problems for which no polynomial time algorithm is known
- ✓ $A \subseteq \Sigma^*$ is *polynomial time reducible* to B , $B \subseteq \Gamma^*$, denoted
- ✓ $A \leq_m^p B$, if there exists a polynomial time computable function $f: \Sigma^* \rightarrow \Gamma^*$ s.t.

$$x \in A \Leftrightarrow f(x) \in B \quad \forall x \in \Sigma^*$$
- ✓ $A \subseteq \{0, 1\}^*$ is *NP-complete*, if
 1. $A \in NP$ and
 2. $B \leq_m^p A$ for all $B \in NP$
- ✓ All problems in NP are polynomial time reducible to a NP -complete problem.
- ✓ If any NP -complete problem is in P , then $P = NP$



- ✓ Showing that $A \in NP$ is NP -complete:
 1. Select a similar problem B that is known to be NP -complete
 2. Give a polynomial time reduction $f: B \leq_m^p A$; by Theorem 7.36 also A is NP -complete
- ✓ NP -complete problems: SAT, CSAT, 3SAT, VC, IS, CLIQUE, Hamiltonian path, TSP, Subset-sum, and minSC

$$PSPACE = \bigcup_{k \geq 0} DSPACE(n^k)$$

$$NPSPACE = \bigcup_{k \geq 0} NSPACE(n^k)$$

$$P \subseteq NP \subseteq \left\{ \begin{array}{l} PSPACE \\ NPSPACE \end{array} \right\} \subseteq EXPTIME \subseteq$$

$$NEXPTIME \subseteq \left\{ \begin{array}{l} EXPSPACE \\ NEXPSPACE \end{array} \right\}$$



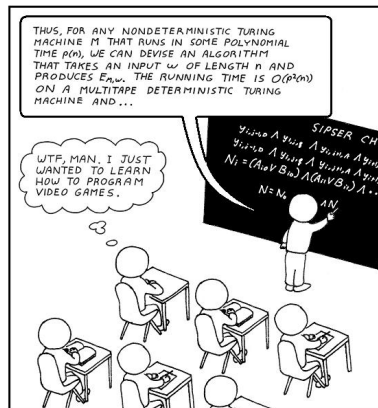
- ✓ TQBF is PSPACE-complete
- ✓ So are the "asymptotic" versions of chess and GO

$$L = DSPACE(\log n)$$

$$NL = NSPACE(\log n)$$

- ✓ $PATH \in NL$ is NL-complete
- ✓ $L \stackrel{?}{=} NL$
- ✓ $NL \subseteq P$
- ✓ One can try to approximate an intractable problem can efficiently
- ✓ Vertex cover has an efficient 2-approximation algorithm
- ✓ Minimum set cover has an efficient $(\ln m + 1)$ -approximation algorithm

THE END



Keep in mind that programming video games also requires understanding computationally demanding problems ...