AAD Aflevering 4

Andreas Hammer (qgk998), Anders Munkvad (tqs326), Johan Topp (fpg662)

December 2023

NP-Completeness

34.4-6

If we have an algorithm which takes n binary arguments $[x_1, x_n]$as input, it would be possible to ask the "decide formula satisfiability" algorithm if the input $x_1 = 0$ has a formula satisfiability which the "decide formula" algorithm can run in polynomial time.
If the result is "no" then any viable solution must belong to $x_1 = 1$. This operation can be continued n times, resulting in a final potential result.
The final bit sequence is a potential solution, but if for all $[x_1, x_n$ the answer has been "no" the final result will also be needed to verify, since it might be a "no", in the case that there is no viable assignment for the problem.

34.5-1

A certificate to this problem would be some mapping of vertexes in G_1 to a subset of vertexes in G_2. Given such a mapping we could pretty easily (and in polynomial time), verify these graphs were isomorphic. This shows that subgraph-isomorphism is $\in NP$, but to show it is NP complete we to find some polynomial time algorithm s.t. we can reduce another NP-complete problem to the subgraph-isomorphism problem.
The clique problem states that a yes instance is if there exists a k-clique in G and no if not. We can in polynomial time make a graph G' with k vertexes which is fully connected, meaning every vertex is connected to every other vertex. If there exists a k-clique size clique in G, meaning it is a yes instance, then there must exist a mapping of the all vertexes in G' into G, since the subgraph of G must also be fully connected since it is a clique, then G' must be an isomorphic subgraph of G, and thus it is also a yes instance of the subgraph-isomorphism problem.
In the case that it is a no instance, meaning there is not a k-clique in G. But since G' is itself a k-clique, it cannot be an isomorphic subgraph of G and thus it must also be a no instance. This concludes the reduction and shows that subgraph-isomorphism problem is NP-complete.

34.5-6

A certificate to the hamiltonian path problem would be a set of vertexes in order of the path. We can easily verify if a set of vertexes is a hamiltonian path by checking if every vertex appears exactly once and there is an edge between every vertex in the certificate. This can easily computed in polynomial time.

We want to reduce hamiltonian cycle to the hamiltonian path problem. For a hamiltonian cycle we can treat any single vertex on the cycle as both the start and endpoint of a path. For an input graph $G(V, E)$ we can select a random vertex $v_i \in V$. We now make a new graph G' where we add a new vertex v_i', which has all the same edges as v_i. Additionally we add a two vertexes a, b, and two edges $(a, v_i), (b, v_i')$. This forces a and b to be the start and endpoint of a Hamiltonian path. This transformation at max has to clone $O(|E|)$ edges, and make two new vertexes, which

can be done in polynomial time.
If there now exists a HAM-path in G' it means there is a path starting and ending in a and b. That means if we merge v_i and v_i' they are both the start and endpoint of a path, meaning we have a cycle.
If there does not exist a HAM-path in G' there cannot be a cycle going through v_i, meaning G is not hamiltonian.
This concludes the reduction and shows that HAM-PATH is NP-complete.

34.5-7

The related decision problem would be possible to model as determining if a graph G contains a cycle of size k.
In order to show that the problem is NP complete, there are two parts to it; 1: showing that a solution can be validated in polynomial time and 2: show that a NP complete problem can be reduced to this decision problem.
Validation that a solution containing k vertices, can be done by validating that each edge exist in the graph, which can be done in $O(|V|)$ since the number of edges are equal to the number of vertices in the graph + 1.
In order to show that LSC is NP-complete, we need to show that another NP-complete problem can be reduced to it. This can be done using the Hamiltonian cycle.
The Hamiltonian cycle is a Hamiltonian path that visits every vertex once, and from the end point, there is a single step to reach the starting node. Thereby the Hamiltonian cycle is equivalent to LCS of max-size.If the graph G contains a LSC of size $|V|$, then there exist a Hamiltonian cycle, if the graph does not contain a LCS of size $|V|$ there are no Hamiltonian cycle.

Exact Exponential Algorithms and Parameterized Complexity

Exercise 1

During the execution of the TSP algorithm described in Fomin-Kratsch, we have that for each $i \in \{2, 3, ..., n$ and $j \in \{1, 2, ..., n-1\}$, we have to keep the values of $OPT[S, c_i]$ for the sets of size j and $j+1$. Hence, the space needed is $\theta(2^n)$. (Fomin-Kratsch, p. 6).

Exercise 2

a

In order to do the induction step for s = 1, substitution is done in to (2).

$$T(n) \leq \left(\frac{1}{3^{\frac{1}{3}}}\right) \cdot 2 \cdot 3^{\frac{n}{3}} + 1 - 1 = \frac{1}{3^{\frac{1}{3}}} \cdot 2 \cdot 3^{\frac{n}{3}}$$

We want to show that the above expression is $\leq 2 \cdot 3^{\frac{n}{3}} - 1$
For $n = 2$ then $2 \cdot 3^{\frac{2-1}{3}} \leq 2 \cdot 3^{\frac{2}{3}} - 1 \Leftrightarrow 2.88 \leq 3.16$ which is the base case.
The induction step:
Assuming that the hypothesis hold for n, the we need to show that it holds for $n+1$ as well.

$$T(n+1) \leq 1 + s \cdot T(n+1-s)$$

Substitute with $s = 1$:

$$T(n+1) \leq 1 + T(n)$$

Inserting boundary for n:

$$\frac{1}{3^{\frac{1}{3}}} \cdot 2 \cdot 3^{\frac{n+1}{3}} \leq 1 + 2 \cdot 3^{\frac{n}{3}} - 1$$

We can rewrite $3^{\frac{n+1}{3}} = 3^{\frac{n}{3}} \cdot 3^{\frac{1}{3}}$, so the statement becomes,

$$2 \cdot 3^{\frac{n}{3}} \leq 2 \cdot 3^{\frac{n}{3}} - 1 + 1$$

,
Which proves that by induction over n that this bound holds when $s = 1$.

b

Now we want to show for the case $s > 1$. For the inequality,

$$T(n) \leq \frac{s}{3^{\frac{s}{3}}} \cdot 2 \cdot 3^{\frac{n}{3}} + 1 - s$$

We know from the lemma that $(s/3^{s/3}) \leq 1$. Since $n \geq 2$, then the expression must always be positive. Meaning that multiplying it by a term less than 1 will make it smaller, so omitting this term means the bound must still hold. We then have.

$$T(n) \leq \frac{s}{3^{\frac{s}{3}}} \cdot 2 \cdot 3^{\frac{n}{3}} + 1 - s \leq 2 \cdot 3^{\frac{n}{3}} + 1 - s$$

If s is larger than one and it must be an integer then $s \geq 2$. If s is larger than 2 we only subtract more from the bound, making it smaller. So setting $s = 2$, must make the statement as large as possible meaning we can bound.

$$T(n) \leq 2 \cdot 3^{\frac{n}{3}} - 1$$

We arrive at the bound we set out to prove.

3

The spacial complexity of the algorithm for the first call to the algorithm (without recursion into account) has a spacial locality bounded by n since the graph contains n edges, When selecting a minimum degree vertex in G at worst case only 2 vertices are removed, so the recursive call will be called with a graph containing $n - 2$ vertices. The number of recursive calls are bounded by the depth of the search tree, and since 2 vertices are removed each time, the number of recursive calls can at max be $\frac{n}{2}$. And since the number of edges removed can at most be $2\sqrt{n}$. So the total complexity will be $f(G) = O(n \cdot n) = O(n^2)$

Exercise 4

a

If there is a solution for a given k then we can find U with at most k^2 vertices. This U can be found in $O(m+n)$ time, and the time to check whether U is indeed a vertex cover of G can be done in $O(k^2)$ time. Therefore, the total running time, for a fixed k, is $O(m+n+k^{2k+2}) \subseteq O_k(m+n)$.

b

Since we want the running time of $O(m + n + 2^k k^2)$ and we use the **BFP-Kernel(k, G)**, given by the lecture, and assume that it runs in $O(m+n)$ when G has m edges and n vertices. In order to do this we realize that the k-vertex cover is simply the complement of the independent set. Namely, if V' is an independent set of $G = (V, E)$ then $V - V'$ is a vertex cover of G. Since we need to check for every edge given in the graph by **BFP-Kernel(k, G)** if it is in the independent set, and this can be done in $n^2 2^n$, or $2^k k^2$, by a naive brute-force algorithm [1], we get the running time of $O(m + n + 2^k k^2)$.

[1] `https://en.wikipedia.org/wiki/Independent_set_(graph_theory)`

1 Andreas Hammer

1.1 NP-completeness

- clear overview of definitions Σ, L, P, NP
- reductions and polynomial time reductions
- Extended NP definitions (NP-hard, NP-Complete, co-NP)
- Show Circut-SAT is NP-complete
- Show how reducing Circut-SAT to SAT implies NP-complete

1.2 Exact Exponential algorithms and parameterized complexity

- What are Exact exponential and parameterized algorithms
- O star notation
- TSP with dynamic programming, time complexity
- Kernelization and vertex cover runtime proof with parameter 'k'

2 Anders Munkvad

2.1 NP-completeness

- Outline: What are hard problems?
- Definition of NP and reduction.
- Definition of NP-completeness.
- Show NP-completeness for BFP (k-vertex cover).
- Show NP-completeness for TSP.

2.2 Exact Exponential algorithms and parameterized complexity

- Motivation: Why do even care about exponential algorithms?
- Example using the Travelling-Salesman problem (Exact Exponential algorithms).
- Example using k-vertex cover (parameterized complexity)

3 Johan Topp

3.1 NP-completeness

- Explain P and NP
- $P \subseteq NP$ (not proved, but assumed)
- Solving NP time vs verifying solution
- NP-hard, NP-complete
- Cook-Levin theorem
- Example of NP-complete: Clique and K-SAT
- Reduction from Clique K-sat

3.2 Exact exponential algorithms and parameterized complexity

- Exponential algorithmns, what kind of problems
- Settle for 2 out of 3(exact,parameterized,approximation).
- O* notation, hiding the polynomial parts
- Runtime of TSP through bruteforce vs dynamic programming.
- Parameterization. Bar fight problem(k-vertex),
- Show complexity of Bar fight problem