Exercise X, Theory of Computation 2025

These exercises are for your own benefit. Feel free to collaborate and share your answers with other students. Solve as many problems as you can and ask for help if you get stuck for too long.

These problems are taken from various sources at EPFL and on the Internet, too numerous to cite individually.

1 A formula is in Disjunctive Normal Form (DNF) if it is an OR ($\vee$) of a number of terms, where each term is an AND ($\wedge$) of literals, as for example $(x \wedge y \wedge \bar{z}) \vee (\bar{y} \wedge z) \vee (\bar{x} \wedge \bar{y})$. Define

$$\text{DNF-SAT} = \{\langle\varphi\rangle : \varphi \text{ is a DNF formula and } \varphi \text{ is satisfiable}\}.$$

1a Prove that DNF-SAT is in **P**.

1b What is wrong with the following reasoning?

Suppose we are given a 3CNF formula, and we want to know if it is satisfiable. We can repeatedly use the distributive law

$$(\varphi_1 \vee \varphi_2) \wedge \psi \equiv (\varphi_1 \wedge \psi) \vee (\varphi_2 \wedge \psi)$$

to construct an equivalent DNF formula. For instance,

$$(x \vee y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y}) \equiv (x \wedge \bar{x}) \vee (x \wedge \bar{y}) \vee (y \wedge \bar{x}) \vee (y \wedge \bar{y}) \vee (\bar{z} \wedge \bar{x}) \vee (\bar{z} \wedge \bar{y}).$$

Then, using the algorithm from Part (a) we can determine, in polynomial time, whether the resulting DNF formula is satisfiable. We have just solved 3SAT in polynomial time. Since 3SAT is **NP**-complete, we conclude that $\mathbf{P} = \mathbf{NP}$.

Solution:

1a On its own, any single clause is satisfiable if and only if it does not contain both a variable and its negation. If it does, then the clearly the clause cannot be satisfied. Else, every literal in the clause can just be set to true. Since the clauses are connected by OR, it is enough to satisfy just one clause. Thus, our algorithm is as follows: For each clause in the formula determine whether it is satisfiable or not according to the above rule. If at least one is, we return yes, if none are, we return no.

1b The reduction is correct, however, it does not run in polynomial time. In fact, in the general case, when we have n clauses of 3 variables each, applying the distributive law repeatedly yields a DNF with 3^n clauses. When we call upon the algorithm from part 1, it will have runtime polynomial in 3^n, which is exponential in n.

2 Prove that the following language is $\mathbf{NP}$-complete:

$$L = \{\langle G = (V, E), k\rangle : \text{there exist disjoint } S_1, S_2 \subseteq V, \text{ such that } |S_1|, |S_2| \geq k, \\ S_1 \text{ is an independent set in } G \text{ and } S_2 \text{ is a clique in } G\}.$$

Solution: We first prove the problem is in $\mathbf{NP}$ and then we prove it in $\mathbf{NP}$-hard by a reduction from Clique which is known to be $\mathbf{NP}$-complete. We first construct a verifier to show $L \in \mathbf{NP}$.

```
On input $((G, k), (S_1, S_2))$, do:
  1. If $|S_1| < k$ or $|S_2| < k$, reject.
  2. If $S_1$ is not an independent set, reject.
  3. If $S_2$ is not a clique, reject.
  4. Accept.
```

Note that this algorithm runs in polynomial time. Moreover, input $((G, k), (S_1, S_2))$ accepts if and only if $(G, k) \in L$. It remains to show that L is $\mathbf{NP}$-hard. We show this by reducing from Clique. Recall that an instance of Clique is a graph G and an integer k. Consider the algorithm f that takes as input (G, k) and outputs (G', k), where G' is the graph obtained by adding k isolated vertices to G. It is clear that f is a polynomial-time algorithm. We claim that $(G, k) \in$ Clique if and only if $f(G, k) = (G', k) \in L$.

- Assume that $(G, k) \in$ Clique: Then there exists a clique of size at least k in G and thus also in G'. The k added isolated vertices in G' form an independent. Therefore, $(G', k) \in L$.
- Assume that $f(G, k) \in L$: Then there exists a clique of size at least k in G', which is also a clique of the same size in G. Thus, $(G, k) \in$ Clique.

3 Recall that a *matching* in an undirected graph $G = (V, E)$ is a subset $M \subseteq E$ such that no two edges in M share an endpoint. We say that a matching M is *well separated* if for any distinct pair of edges $e, e' \in M$ each endpoint of e is at distance at least 2 from each endpoint of e'. Prove that the following problem is $\mathbf{NP}$-complete:

$$\textsc{SepMatch} = \{\langle G, k\rangle : G \text{ contains a well separated matching of size } k\}$$

Solution: Note first that $\textsc{SepMatch} \in \mathbf{NP}$ because a certificate is simply a subset of edges $M \subseteq E$ (this has polynomial size) which is well-separated and has size k (this can be checked in polynomial time).

We further show that $\textsc{IndSet} \leq_p \textsc{SepMatch}$. Let $\langle G = (V, E), k\rangle$ be an independent set instance. Consider the graph $G' = (V', E')$ which is a copy of G but where each original vertex $v \in V$ has a new *dual* vertex v' attached to it (see Figure 1 for an example). This transformation can clearly be performed in polynomial time. We claim that

$$\langle G, k\rangle \in \textsc{IndSet} \iff \langle G', k\rangle \in \textsc{SepMatch}.$$

Suppose first that G has an independent set S of size k: The set of edges $M = \{vv' : v \in S\}$ is a well separated matching of size k for G'. Indeed, pick any two edges uu' and vv' of M. Note that the two closest endpoints are u and v and because S is an independent set. they are at distance at least 2.

On the other hand, suppose that G' has a well separated matching M of size k. We may assume that M only uses edges of type vv'. Indeed, for any other type of edge, we can replaced one of its endpoints accordingly, without violating the well-separation. Now, $S = \{v : vv' \in M\}$ is an independent set of size k for G. Indeed, if $u, v \in S$ are adjacent, then M was not well separated to begin with.

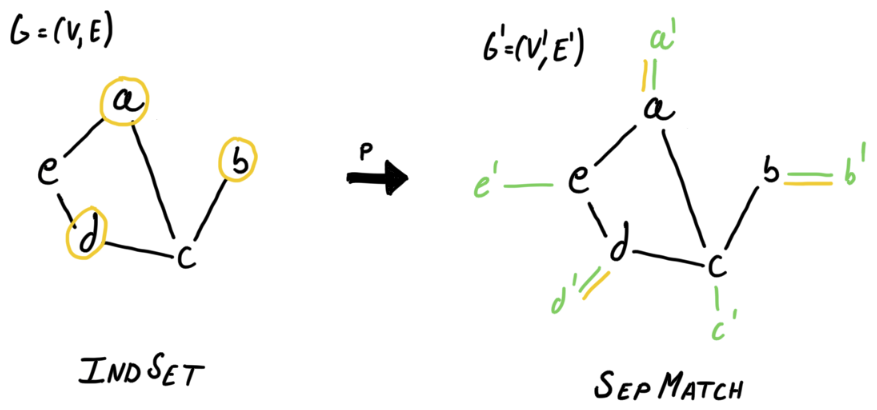

Figure 1. Transforming an IndSet instance into a SepMatch one. Yellow vertices (resp. edges) are maximum independent set (resp. well-separated matching).

4 Prove that the following problem is **NP**-hard: Given n integers $a_1, a_2, \dots, a_n$, is there a set $S \subseteq \{1, 2, \dots, n\}$ such that

$$\sum_{i \in S} a_i = \sum_{i \notin S} a_i?$$

Solution: Let us denote this problem by L (it is often referred to as the partition problem). We show that L is **NP**-hard by reducing from SubsetSum. Recall that an instance to SubsetSum consists of a set S and an integer M. Consider the algorithm f, which on input (S, M) outputs

$$S' = S \cup \left\{ \sum_{s \in S} s - 2M \right\}.$$

It is clear that f runs in polynomial time. We how show that in fact $(S, M) \in$ SubsetSum if and only if $f(S, M) = S' \in L$.

- Assume that $(S, M) \in$ SubsetSum: Then there exists $T \subseteq S$ such that $\sum_{s \in T} s = M$. Consider $T' = T \cup \{\sum_{s \in S} s - 2M\}$. We observe that $\sum_{s \in T'} s = M + \sum_{s \in S} s - 2M = \sum_{s \in S} s - M$, and $\sum_{s' \in S' \setminus T'} = (\sum_{s \in S} s) - M$. Therefore, $S' \in L$ as desired.
- Assume that $S' \in L$: Then there exists $T \subseteq S'$ such that

$$\sum_{s \in T} s = \sum_{s \in S' \setminus T} s = \frac{\sum_{s \in S} s + \sum_{s \in S} s - 2M}{2} = \sum_{s \in S} s - M.$$

One of T or $S' \setminus T$ contains the number $\sum_{s \in S} s - 2M$, and removing it gives us a set of numbers in S whose sum is M. Therefore, $(S, M) \in$ SubsetSum as desired.