Algorithm Design and Analysis (Fall 2021)
Assignment 6

1. (50 Points) Consider the following variant of the scheduling problem. We have n jobs with size (time required for completion) $p_1, \ldots, p_n \in \mathbb{Z}^+$. Instead of fixing the number of machines m and minimizing the makespan, we fix the makespan and minimize the number of machines used. That is, you can decide how many (identical) machines to use, but each machine can be operated for at most $T \in \mathbb{Z}^+$ units of time. Assume $p_i \leq T$ for each $i = 1, \ldots, n$. Your objective is to minimize the number of machines used, while completing all the jobs.

   (a) (20 Points) Show that this minimization problem is NP-hard.

   (b) (20 Points) Consider the following local search algorithm. Initialize the solution where n machines are used such that each machine handles a single job. Do the following update to the solution until no more update is possible: if there are two machines such that the total size of the jobs on the two machines is less than T, update the solution by using only one machine to complete all these jobs (instead of using two machines). Show that this algorithm gives a 2-approximation.

   (c) (10 Points) Provide a tight example showing that the algorithm in (b) can do 1.5-approximation at best.

   (a) This optimization problem is called *the bin packing problem*. We will show that it is NP-hard to decide if we can complete all the jobs by at most *two* machines. We reduce this decision problem from Partition+.

   Given a Partition+ instance $S = a_1, \ldots, a_n$, we construct an instance $(p_1, \ldots, p_n, T)$ of the bin packing problem as follows:

   - for each $i = 1, \ldots, n$, $p_i = a_i$;
   - set $T = \lfloor \frac{1}{2} \sum_{i=1}^{n} a_i \rfloor$.

   If the Partition+ instance is a yes instance, we can partition S to S_1 and S_2 such that $\sum_{a_i \in S_1} a_i = \sum_{a_i \in S_2} a_i = \frac{1}{2} \sum_{i=1}^{n} a_i$. For the bin packing instance, we can load all the jobs corresponding to S_1 to the first machine, and all the jobs corresponding to S_2 to the second machine. Each machine needs exactly time T, so this is a valid job assignment. We can complete all the jobs with at most two machines.

   Suppose we only need at most two machines for the bin packing instance. Let T_1 be the set of jobs on the first machine, and T_2 be the set of jobs on the second machine (set $T_2 = \emptyset$ if only one machine is needed). We have $\sum_{i \in T_1} p_i \leq T$ and $\sum_{i \in T_2} p_i \leq T$. Since T_1, T_2 is a partition of all jobs and $\sum_{i=1}^{n} p_i = \sum_{i=1}^{n} a_i = 2T$, we must have $\sum_{i \in T_1} p_i = T$ and $\sum_{i \in T_2} p_i = T$. Let S_1, S_2 be the two subsets in the Partition+

instance corresponding to T_1, T_2 respectively. It is straightforward that $\sum_{a_i \in S_1} a_i = T$ and $\sum_{a_i \in S_2} a_i = T$. Thus, the PARTITION+ instance is a yes instance.

In fact, we can prove a stronger result than just the NP-hardness.

If P $\neq$ NP*, the bin packing problem do not admit a polynomial-time* $\left(\frac{3}{2} - \varepsilon\right)$*-approximation algorithm for any* $\varepsilon > 0$*.*

Proof. Suppose we have a polynomial-time $\left(\frac{3}{2} - \varepsilon\right)$-approximation algorithm for any $\varepsilon > 0$. We show that there exists a polynomial-time algorithm for PARTITION+. Given a PARTITION+ instance, we convert it to the bin packing instance by the above-mentioned construction. The construction can clearly be done in polynomial time. Then we run the $\left(\frac{3}{2} - \varepsilon\right)$-approximation algorithm for the obtained bin packing instance. If the output of the algorithm uses at most two machines, we know the PARTITION+ instance is a yes instance. If the output of the algorithm uses at least three machines, we know the optimal number of machines is at least $\frac{1}{3/2-\varepsilon} \times 3 > 2$. Since the number of machines is an integer, we know the optimal solution will use at least three machines, and the PARTITION+ instance should be a no instance. This gives us a polynomial time algorithm for PARTITION+. Since PARTITION+ is NP-complete, it is a contradiction to P $\neq$ NP. □

(b) The important observation here is that, for the output of the local search algorithm, the total time for jobs on every two machines is larger than T.

Let k be the number of machines the algorithm uses, and k^* be the optimal number of machines. Let m_i be the total size of jobs on machine i for the output of the local search algorithm. We have $m_i + m_j > T$ for any $1 \leq i, j \leq k$ by our observation.

On the one hand, we have

$$\sum_{i=1}^{k} m_i \leq k^* T,$$

as $\sum_{i=1}^{k} m_i$ is the overall size of all jobs and these jobs need to be contained in those k^* machines in the optimal solution.

On the other hand, since $m_i + m_j > T$, we have

$$2\sum_{i=1}^{k} m_i = (m_1 + m_2) + (m_2 + m_3) + \cdots + (m_{k-1} + m_k) + (m_k + m_1) > kT.$$

Combining the two inequalities, we have

$$\frac{\sum_{i=1}^{k} m_i}{k^*} \leq T < \frac{2\sum_{i=1}^{k} m_i}{k},$$

which implies $k < 2k^*$.

(c) A tight example could be $(p_1, p_2, p_3, p_4) = (1, 1, 2, 2)$ and $T = 3$. The optimal solution uses two machines with p_1, p_3 on the first and p_2, p_4 on the second. If we use the local search algorithm and first combine the machine containing p_1 and the machine containing p_2, it is easy to check that at least three machines are required.

2. (50 Points) Choose *any one* of the following questions. (You are encouraged to solve as many the remaining questions as possible "in your mind".)

   (a) Given an undirected graph $G = (V, E)$ with $n = |V|$, decide if G contains a clique with size exactly $n/2$. Prove that this problem is NP-complete.

   (b) Given an undirected graph $G = (V, E)$, the *3-coloring* problem asks if there is a way to color all the vertices by using three colors, say, red, blue and green, such that every two adjacent vertices have different colors. Prove that 3-coloring is NP-complete.

   (c) Given two undirected graphs G and H, decide if H is a subgraph of G. Prove that this problem is NP-complete.

   (d) Given an undirected graph $G = (V, E)$ and an integer k, decide if G has a spanning tree with maximum degree at most k. Prove that this problem is NP-complete.

   (e) Given a ground set $U = \{1, \ldots, n\}$ and a collection of its subsets $\mathcal{S} = \{S_1, \ldots, S_m\}$, the *exact cover* problem asks if we can find a subcollection $\mathcal{T} \subseteq \mathcal{S}$ such that $\bigcup_{S \in \mathcal{T}} S = U$ and $S_i \cap S_j = \emptyset$ for any $S_i, S_j \in \mathcal{T}$. Prove that exact cover is NP-complete.

   (f) Given a collection of integers (can be negative), decide if there is a subcollection with sum exactly 0. Prove that this problem is NP-complete.

   (g) In an undirected graph $G = (V, E)$, each vertex can be colored either black or white. After an initial color configuration, a vertex will become black if all its neighbors are black, and the updates go on and on until no more update is possible. (Notice that once a vertex is black, it will be black forever.) Now, you are given an initial configuration where all vertices are white, and you need to change k vertices from white to black such that all vertices will eventually become black after updates. Prove that it is NP-complete to decide if this is possible.

I am going to start from the easiest to the hardest.

(c) Difficulty: $*$

The problem is clearly in NP, as the set of vertices in G that form the subgraph H is a certificate.

To show it is NP-complete, we reduce it from CLIQUE. Given a CLIQUE instance $(G = (V, E), k)$, the instance (G', H') of this problem is constructed as $G' = G$ and H' is a complete graph with k vertices. It is straightforward to check that $(G = (V, E), k)$ is a yes instance if and only if (G', H') is a yes instance.

(d) Difficulty: $*$

The problem is clearly in NP, as the set of edges forming the spanning tree with maximum degree at most k is a certificate.

To show it is NP-complete, we reduce it from HamiltonianPath. Given a HamiltonianPath instance $G = (V, E)$, the instance (G', k) of this problem is constructed as $G' = G$ and $k = 2$.

If $G = (V, E)$ is a yes HamiltonianPath, there is a Hamiltonian path in G, and this is also a spanning tree with maximum degree 2, so (G', k) is a yes instance.

If (G', k) is a yes instance, then a spanning tree with maximum degree 2 must also be a Hamiltonian path, so $G = (V, E)$ is a yes HamiltonianPath instance.

(f) Difficulty: $*$

The problem is clearly in NP, as the subcollection of integers with sum 0 is a certificate.

To show it is NP-complete, we reduce it from SubsetSum+. Given a SubsetSum+ instance (S, k), the instance S' of this problem is constructed as $S' = S \cup \{-k\}$.

If (S, k) is a yes SubsetSum+ instance, there exists a subcollection T of S with sum k. Then adding $-k$ to T gives a subcollection of S' with sum 0, which means S' is also a yes instance.

If S' is a yes instance, the subcollection T' with sum 0 must contain $-k$, as $-k$ is the only negative number in S'. Then excluding $-k$ from T' gives a subcollection of S with sum k, which means (S, k) is a yes SubsetSum+ instance.

(a) Difficulty: $**$

The problem is clearly in NP, as the set of $n/2$ vertices that form a clique is a certificate.

To show that the problem is NP-complete, we reduce it from Clique. Given a Clique instance $(G = (V, E), k)$, we construct the following half-clique instance G':

- If $k < \frac{|V|}{2}$, G' is obtained by adding $|V| - 2k$ extra vertices, connect those extra vertices to each other, and then connect each extra vertex to all vertices in V.
- If $k = \frac{|V|}{2}$, $G' = G$.
- If $k > \frac{|V|}{2}$, G' is obtained by adding $2k - |V|$ extra *isolated* vertices.

It is straightforward to check that G has a k-clique if and only if G' has a $\frac{n}{2}$-clique. The details are omitted here.

(g) Difficulty: $**$

The problem is clearly in NP, as the set of vertices whose colors are initially changed to black is a certificate.

To show it is NP-complete, we reduce it from VertexCover. Here is an important observation. Let S be the subset of vertices whose colors are changed to black initially. We will prove that all vertices will eventually turn to black *if and only if* S is a vertex cover of this graph.

If S is a vertex cover, then for each white vertex u, all its neighbors must be black, or in other words, in S. Otherwise, if a neighbor v of u is not in S, then (u, v) is not covered by S and S cannot be a vertex cover. Given that all the neighbors for each white vertex are black, each white vertex will become black in the next update.

If S is not a vertex cover, then there exists an edge (u, v) that is not covered by S. This means u and v are both white. By our rule of update, u and v can never become black: it is never possible that all neighbors of u are black, nor it is possible for v.

Therefore, the reduction is simple. For the VertexCover instance (G, k), the instance for this problem is also (G, k). The remaining details are omitted.

(b) Difficulty: $***$

[Solution 1: Textbook Solution]

3-coloring is clearly in NP, as a color assignment of all vertices is a certificate for a yes instance. To show it is NP-complete, we reduce it from 3-SAT.

Given a 3-SAT instance ϕ, we construct a 3-coloring instance $G = (V, E)$ as follows. We will name the three colors T, F, N which stand for "true", "false", "neutral". The reason for this naming will be apparent soon. Firstly, we construct three vertices named t, f, n and three edges $\{t, f\}, \{f, n\}, \{n, t\}$. It is easy to see these 3 vertices, forming a triangle, must be assigned different colors. Without loss of generality, we assume $c(t) = T, c(f) = F, c(n) = N$, where $c : V \to \{T, F, N\}$ is the color assignment function. We call this triangle the "palette": in the remaining part of the graph, whenever we want to enforce that a vertex cannot be assigned a particular color, we connect it to one of the three vertices in the palette.

For each variable x_i in ϕ, we construct two vertices $x_i, \neg x_i$, and three edges $(x_i, \neg x_i)$, (x_i, n), $(\neg x_i, n)$. This ensures that it must be either $c(x_i) = T, c(\neg x_i) = F$ or $c(x_i) = F, c(\neg x_i) = T$. The former case corresponds to assigning `true` to variable x_i, and the latter case corresponds to assigning `false` to variable x_i.

After we have constructed the gadgets simulating the Boolean assignment for variables, we need to create a gadget to simulate each clause m. We use $m = (x_i \vee \neg x_j \vee x_k)$ as an example to illustrate the reduction. Firstly, we create a vertex m and connect it to the two vertices n, f, so that we can only have $c(m) = T$ (i.e., the clause must be evaluated to `true`). Then, we need to create a gadget that connects the three vertices x_i, $\neg x_j$ and x_k (constructed in the previous step) as "inputs", and connects vertex m at the other end as "output". The gadget must simulate the logical OR operation.

The gadget shown on the left-hand side in Fig. 1 simulates the logical OR operation for two inputs: if both two input vertices are assigned F, then the output vertex cannot be assigned T, for otherwise there is no valid way to assign colors for vertices A and B; if at least one input vertex is assigned T, say, Input 1 is assigned T, then it is possible to assign the output vertex T, since it is always valid to assign F to A and N to B. The gadget for the clause $m = (x_i \vee \neg x_j \vee x_k)$ can then be constructed by applying two OR gadgets, shown on the right-hand side of Fig. 1. We do this for all the clauses.

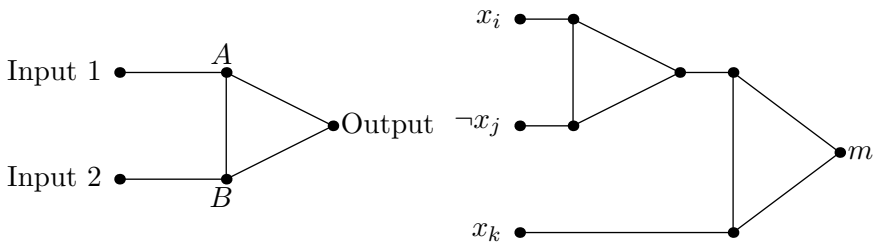

Figure 1: The OR gadget (left-hand side) and the gadget for the clause $m = (x_i \vee \bar{x}_j \vee x_k)$ (right-hand side).

The remaining part of the proof is straightforward. We have shown that G simulate assignments to ϕ. Thus, ϕ is satisfiable if and only if there is a valid color assignment to vertices in G. Finally, it is easy to verify that the construction of G can be done in a polynomial time.

[Solution 2: Wenqian Wang's Solution]

The proof for 3-coloring is in NP is the same as before. To show it is NP-complete, we introduce an intermediate problem NotAllEqual-3SAT.

Problem 1 (NotAllEqual-3SAT)**.** Given a 3-CNF Boolean formula ϕ, decide if there is an assignment such that each clause contains at least one true literal and one false literal.

Lemma 2. NotAllEqual-3SAT *is* NP*-complete.*

Proof. The problem is clearly in NP, and we will present a reduction from 3SAT to NotAllEqual-3SAT. Given a 3SAT instance ϕ, we construct a NotAllEqual-3SAT instance ϕ' as follows. Firstly, introduce a new Boolean variable w. Then, for each clause $C_j = (\ell_1 \vee \ell_2 \vee \ell_3)$ in ϕ, we introduce two more variables y_j, z_j and use three clauses in ϕ' to represent C_j:

$$(w \vee \ell_1 \vee y_j) \wedge (\neg y_j \vee \ell_2 \vee z_j) \wedge (\neg z_j \vee \ell_3 \vee w). \tag{1}$$

Notice that the variable w is used for all triples of three clauses, while y_j, z_j are only used for the j-th triple.

If ϕ is a yes 3SAT instance, we will show that ϕ' is also a yes instance by showing each of the three clauses in (1) contains a `true` and a `false`. We will set $w =$ `false`. Since ϕ is a yes 3SAT instance, we know at least one of ℓ_1, ℓ_2, ℓ_3 is `true`. If $\ell_1 =$ `true`, we can set $y_j =$ `false` and $z_j =$ `false`. If $\ell_2 =$ `true`, we can set $y_j =$ `true` and $z_j =$ `false`. If $\ell_3 =$ `true`, we can set $y_j =$ `true` and $z_j =$ `true`. In each scenario, it can be check that each of the three clauses in (1) contains a `true` and a `false`. Thus, ϕ' is a yes NotAllEqual-3SAT instance.

If ϕ' is a yes NotAllEqual-3SAT instance, we aim to show that at least one of ℓ_1, ℓ_2, ℓ_3 must be true, which will imply ϕ is a yes 3SAT instance. Firstly, we assume without loss of generality that $w =$ `false`. If $w =$ `true`, we can flip the values for all variables, which will also be a valid assignment. Suppose for the sake of contradiction that $\ell_1 = \ell_2 = \ell_3 =$ `false`. To ensure the first and the third clauses in (1) contain at least one `true`, we need to set $y_j =$ `true` and $z_j =$ `false`. In this case, the second clause does not contain any `true`, which is a contradiction. □

Lemma 3. NotAllEqual-3SAT $\leq_k$ *3-coloring.*

Proof. The main idea here is that a not-all-equal gadget is much easier to construct. The figure on the left-hand side of Fig. 2 demonstrates a not-all-equal gadget.

It is easy to check that, if all the three inputs have the same value, or the same color, the three middle vertices v_1, v_2, v_3 can only choose from two colors, which is impossible. On the other hand, if the three inputs are not all equal, then we can properly choose colors for v_1, v_2, v_3. For example, if Input 1 is T and Input 2 is F, regardless of the value of Input 3, assigning $c(v_1) = F, c(v_2) = T, c(v_3) = N$ is always valid. Thus, this gadget faithfully performs the not-all-equal job.

Owing to the simplicity of the not-all-equal gadget, we do not need a triangle as a palette as before. All we need is a vertex n that is assumed (without loss of generality) to be colored with N. We connect n to the vertex representing x_i and the vertex representing $\neg x_i$. The figure on the right-hand side of Fig. 2 demonstrates these features.

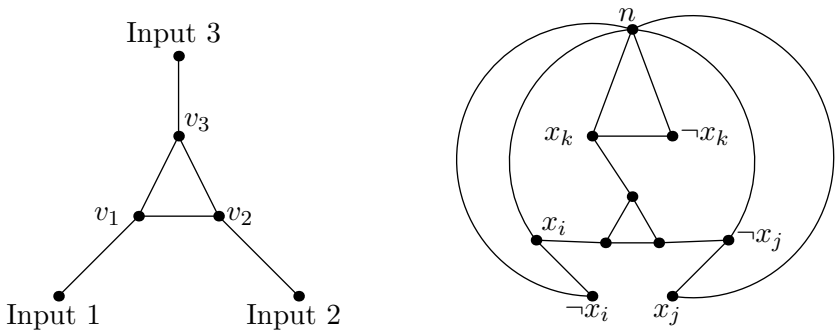

Figure 2: The not-all-equal gadget (left-hand side); the gadget for the clause $m = (x_i \vee \bar{x}_j \vee x_k)$ and how it is connected with other vertices (right-hand side).

The remaining parts of the proof are straightforward, and thus omitted. □

Lemma 2 and Lemma 3 immediately imply 3-coloring is NP-complete.

(e) Difficulty: $***$

[Solution 1: Biaoshuai Tao's Solution]

The problem is clearly in NP, as a subcollection of subsets that exactly covers U is a certificate.

To show it is NP-complete, we reduce it from 3-coloring. Given a 3-coloring instance $G = (V, E)$, we construct an exact cover instance as follows.

The ground set U is constructed as follows.

- For each vertex $u \in V$, we construct an element $e_u \in U$.
- For each edge $(u, v) \in E$, we construct three elements $e^r_{uv}, e^g_{uv}, e^b_{uv} \in U$, where the superscripts r, g, b stand for red, green and blue respectively.

The collection $\mathcal{S}$ is constructed as follows:

- For each vertex $u \in V$, we construct three subsets $S^r_u, S^g_u, S^b_u \in \mathcal{S}$ defined as follows.
  - include $e_u \in S^r_u$
  - include $e^r_{uv} \in S^r_u$ for each u's neighbor v.
  - S^g_u and S^b_u are defined similarly.
- For each edge $(u, v) \in E$, construct three subsets $S^r_{uv}, S^g_{uv}, S^b_{uv} \in \mathcal{S}$ such that they contains $e^r_{uv}, e^g_{uv}, e^b_{uv}$ respectively. Notice that each of $S^r_{uv}, S^g_{uv}, S^b_{uv}$ contains only one elements.

For a demonstrating example, suppose the 3-coloring instance is just a triangle with three vertices u, v, w. We have $U = \{e_u, e_v, e_w, e^r_{uv}, e^g_{uv}, e^b_{uv}, e^r_{vw}, e^g_{vw}, e^b_{vw}, e^r_{wu}, e^g_{wu}, e^b_{wu}\}$. The collection $\mathcal{S}$ contains 18 subsets:

$$S^r_u = \{e_u, e^r_{uv}, e^r_{wu}\}, S^g_u = \{e_u, e^g_{uv}, e^g_{wu}\}, S^b_u = \{e_u, e^b_{uv}, e^b_{wu}\}$$

$$S^r_v = \{e_v, e^r_{uv}, e^r_{vw}\}, S^g_v = \{e_v, e^g_{uv}, e^g_{vw}\}, S^b_v = \{e_v, e^b_{uv}, e^b_{vw}\}$$

$$S^r_w = \{e_w, e^r_{vw}, e^r_{wu}\}, S^g_w = \{e_w, e^g_{vw}, e^g_{wu}\}, S^b_w = \{e_w, e^b_{vw}, e^b_{wu}\}$$

$$S^r_{uv} = \{e^r_{uv}\}, S^g_{uv} = \{e^g_{uv}\}, S^b_{uv} = \{e^b_{uv}\}, S^r_{vw} = \{e^r_{vw}\}, S^g_{vw} = \{e^g_{vw}\}, S^b_{vw} = \{e^b_{vw}\}, S^r_{wu} = \{e^r_{wu}\},$$

$$S^g_{wu} = \{e^g_{wu}\}, S^b_{wu} = \{e^b_{wu}\}.$$

This whole construction can clearly be done in polynomial time.

If the 3-coloring instance is a yes instance, let $c : V \to \{r, g, b\}$ be a valid coloring. We further assign a color to an edge (u, v), denoted by $c(u, v)$, such that $c(u, v)$ is different from $c(u)$ and $c(v)$. Given a valid coloring of vertices, each edge is uniquely colored. For example, if $c(u) = r$ and $c(v) = b$, then we have $c(u, v) = g$. To show the exact cover instance we constructed is a yes instance, we consider the following subcollection $\mathcal{T} \subseteq \mathcal{S}$:

- For each vertex u, include $S^{c(u)}_u$ to $\mathcal{T}$;
- For each edge (u, v), include $S^{c(u,v)}_{uv}$ to $\mathcal{T}$.

We will prove that $\mathcal{T}$ is an exact cover.

For each element e_u corresponding to vertex u in G, it is covered by exactly one of $S^{c(u)}_u$. For each element e^r_{uv} corresponding to edge (u, v) in G, it is covered by S^r_u if $c(u) = r$, it is covered by S^r_v if $c(v) = r$, and it is covered by S^r_{uv} if $c(u, v) = r$. Since exactly one of $c(u) = r, c(v) = r, c(u, v) = r$ can happen, e^r_{uv} is covered by exactly one subset. The same holds for e^g_{uv} and e^b_{uv}. We conclude that the exact cover instance we constructed is a yes instance.

Now, suppose the exact cover instance is a yes instance. We will show that the 3-coloring instance is a yes instance. Let $\mathcal{T}$ be a valid solution to the exact cover instance. Then exactly one of S^r_u, S^g_u, S^b_u is included in $\mathcal{T}$, as these are the three subsets that contains e_u. This corresponds to a coloring $c : V \to \{r, g, b\}$: if S^r_u is included, we assign $c(u) = r$; if S^g_u is included, we assign $c(u) = g$; if S^b_u is included, we assign $c(u) = b$. It remains to show that c is a valid 3-coloring. Suppose for the sake of contradiction that there exists $(u, v) \in E$ with $c(u) = c(v)$. Assume $c(u) = c(v) = r$ without loss of generality. Then we have included both S^r_u and S^r_v in $\mathcal{T}$. In this case, the element e^r_{uv} is covered by both S^r_u and S^r_v, which contradicts to that $\mathcal{T}$ is an exact cover.

[Solution 2: Jiaxin Song's Solution]

The proof that exact cover is in NP is the same as before.

To show it is NP-complete, we reduce it from 3SAT. Given a 3SAT instance ϕ, we construct an exact cover instance as follows.

The ground set U is constructed as follows:

- For each variable x_i, we construct an element $e_{x_i} \in U$.
- For each clause $C_j = (\ell_1 \vee \ell_2 \vee \ell_3)$, we construct four elements $e_{C_j}, e_{C_j}^{\ell_1}, e_{C_j}^{\ell_2}, e_{C_j}^{\ell_3} \in U$.

The collection $\mathcal{S}$ is constructed as follows:

- For each variable x_i, construct two subsets $S_{x_i}^T, S_{x_i}^F \in \mathcal{S}$ where
  - $S_{x_i}^T = \{e_{x_i}\} \cup \{e_{C_j}^{x_i} : x_i \text{ is a literal in } C_j\}$;
  - $S_{x_i}^F = \{e_{x_i}\} \cup \{e_{C_j}^{\neg x_i} : \neg x_i \text{ is a literal in } C_j\}$.
- For each clause $C_j = (\ell_1 \vee \ell_2 \vee \ell_3)$, construct seven subsets $S_{C_j}^{TTT}, S_{C_j}^{TTF}, S_{C_j}^{TFT}, S_{C_j}^{FTT}, S_{C_j}^{FFT}, S_{C_j}^{FTF}, S_{C_j}^{TFF} \in \mathcal{S}$ where
  - $S_{C_j}^{TTT} = \{e_{C_j}\}$
  - $S_{C_j}^{TTF} = \{e_{C_j}, e_{C_j}^{\ell_3}\}$
  - $S_{C_j}^{TFT} = \{e_{C_j}, e_{C_j}^{\ell_2}\}$
  - $S_{C_j}^{FTT} = \{e_{C_j}, e_{C_j}^{\ell_1}\}$
  - $S_{C_j}^{FFT} = \{e_{C_j}, e_{C_j}^{\ell_1}, e_{C_j}^{\ell_2}\}$
  - $S_{C_j}^{FTF} = \{e_{C_j}, e_{C_j}^{\ell_1}, e_{C_j}^{\ell_3}\}$
  - $S_{C_j}^{TFF} = \{e_{C_j}, e_{C_j}^{\ell_2}, e_{C_j}^{\ell_3}\}$

For a demonstrating example, suppose ϕ consist of only one clause $C_1 = (x_1 \vee \neg x_2 \vee x_3)$. The ground set is $U = \{e_{x_1}, e_{x_2}, e_{x_3}, e_{C_1}, e_{C_1}^{x_1}, e_{C_1}^{\neg x_2}, e_{C_1}^{x_3}\}$. The collection $\mathcal{S}$ consists of the following 13 subsets:

$$S_{x_1}^T = \{e_{x_1}, e_{C_1}^{x_1}\}, S_{x_1}^F = \{e_{x_1}\}, S_{x_2}^T = \{e_{x_2}\}, S_{x_2}^F = \{e_{x_2}, e_{C_1}^{\neg x_2}\}, S_{x_3}^T = \{e_{x_3}, e_{C_3}^{x_3}\}, S_{x_3}^F = \{e_{x_3}\}$$

$$S_{C_1}^{TTT} = \{e_{C_1}\}, S_{C_1}^{TTF} = \{e_{C_1}, e_{C_1}^{x_3}\}, S_{C_1}^{TFT} = \{e_{C_1}, e_{C_1}^{\neg x_2}\}, S_{C_1}^{FTT} = \{e_{C_1}, e_{C_1}^{x_1}\}$$

$$S_{C_1}^{FFT} = \{e_{C_1}, e_{C_1}^{x_1}, e_{C_1}^{\neg x_2}\}, S_{C_1}^{FTF} = \{e_{C_1}, e_{C_1}^{x_1}, e_{C_1}^{x_3}\}, S_{C_1}^{TFF} = \{e_{C_1}, e_{C_1}^{\neg x_2}, e_{C_1}^{x_3}\}$$

The whole construction can clearly be done in polynomial time.

Suppose ϕ is a yes 3SAT instance. We will construct an exact cover $\mathcal{T}$ to show that the exact cover instance is a yes instance. For each x_i, if $x_i = \texttt{true}$, we include $S_{x_i}^T \in \mathcal{T}$; otherwise, include $S_{x_i}^F \in \mathcal{T}$. For each clause C_j, we check the values of the three literals, and include the corresponding subset $S_{C_j}^{XXX}$ in $\mathcal{T}$. For example, for the clause

$C_1 = (x_1 \vee \neg x_2 \vee x_3)$, if the assignment is $x_1 = x_2 = x_3 = \texttt{true}$, the first and the third literals are `true` and the second is `false`, and in this case we will include $S_{C_1}^{TFT}$. We will show $\mathcal{T}$ is an exact cover.

Firstly, each "variable element" e_{x_i} is covered exactly once by either $S_{x_i}^T$ or $S_{x_i}^F$. Secondly, each "clause element" e_{C_j} is covered exactly once since we have selected exactly one of those seven $S_{C_j}^{XXX}$. Lastly, for each element $e_{C_j}^{x_i}$, it is covered exactly once: if we have not selected $S_{x_i}^T$, based on our construction of $\mathcal{T}$, it will be covered by the subset $S_{C_j}^{XXX}$ we selected; if we have selected $S_{x_i}^T$, it will not be covered by the subset $S_{C_j}^{XXX}$ we selected. The same analysis holds for each element $e_{C_j}^{\neg x_i}$: it will be covered by either $S_{x_i}^F$ or the subset $S_{C_j}^{XXX}$ we selected, but not both. We conclude that $\mathcal{T}$ is an exact cover.

Now, suppose we have an exact cover $\mathcal{T}$. We will show that ϕ is satisfiable. For each variable x_i, exactly one of $S_{x_i}^T$ and $S_{x_i}^F$ must be selected to cover e_{x_i}. This gives us a natural assignment to $x_1, \ldots, x_n$. We will show that this is a satisfiable assignment. Suppose for the sake of contradiction that there is a clause C_j where the values of all the three literals are `false`. Using the same example $C_1 = (x_1 \vee \neg x_2 \vee x_3)$. Suppose $x_1 = x_3 = \texttt{false}$ and $x_2 = \texttt{true}$. Then we have selected $S_{x_1}^F, S_{x_2}^T, S_{x_3}^F$. The three elements $e_{C_1}^{x_1}, e_{C_1}^{\neg x_2}, e_{C_1}^{x_3}$ are not covered by any of $S_{x_1}^F, S_{x_2}^T, S_{x_3}^F$, so they need to be covered by those $S_{C_1}^{XXX}$. We can only select one of those $S_{C_1}^{XXX}$: if we select more than one of those, the element e_{C_1} will be covered more than once. On the other hand, only one $S_{C_1}^{XXX}$ cannot cover all the three elements $e_{C_1}^{x_1}, e_{C_1}^{\neg x_2}, e_{C_1}^{x_3}$ by our construction (we have not included "$S_{C_1}^{FFF}$" as a subset in $\mathcal{S}$). Therefore, $\mathcal{T}$ cannot be an exact cover, which leads to a contradiction.