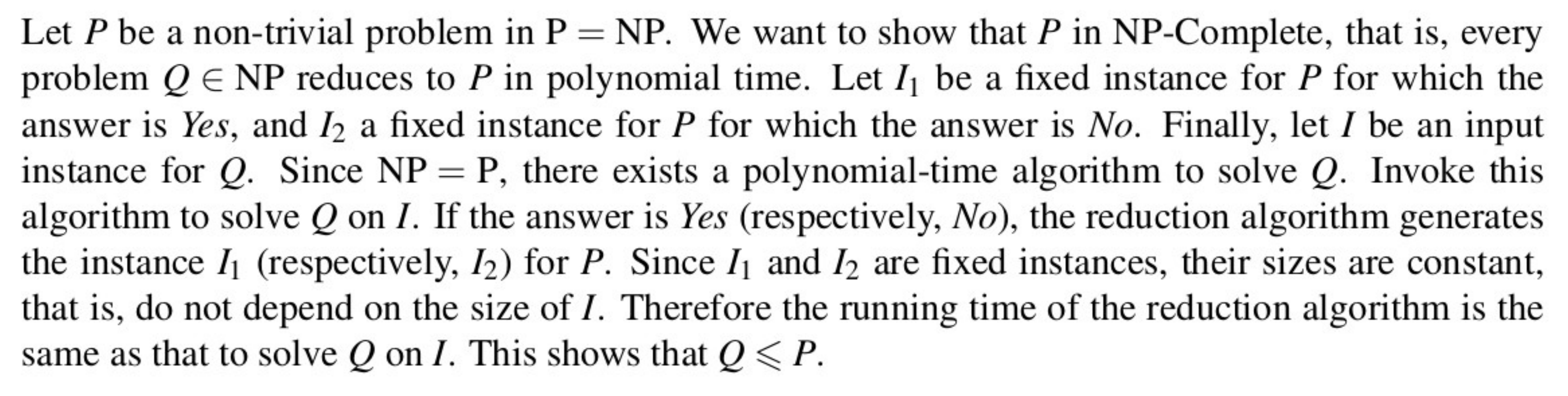
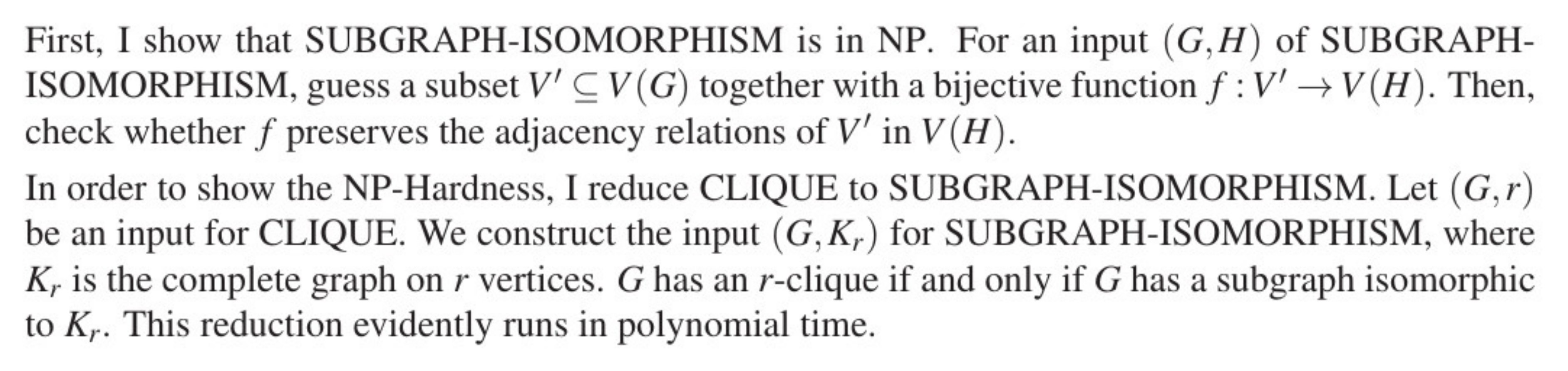
1. Prove that if P = NP, then every non-trivial problem in this class is NP-Complete.



Let P be a non-trivial problem in P = NP. We want to show that P in NP-Complete, that is, every problem $Q \in$ NP reduces to P in polynomial time. Let I_1 be a fixed instance for P for which the answer is *Yes*, and I_2 a fixed instance for P for which the answer is *No*. Finally, let I be an input instance for Q. Since NP = P, there exists a polynomial-time algorithm to solve Q. Invoke this algorithm to solve Q on I. If the answer is *Yes* (respectively, *No*), the reduction algorithm generates the instance I_1 (respectively, I_2) for P. Since I_1 and I_2 are fixed instances, their sizes are constant, that is, do not depend on the size of I. Therefore the running time of the reduction algorithm is the same as that to solve Q on I. This shows that $Q \leqslant P$.

2. Prove that SUBGRAPH-ISOMORPHISM is NP-Complete.

G, G' graphs. Decide whether G contains a subgraph isomorphic to G', that is, whether there exists an injective function f : V' → V such that (u',v') ϵ E' if and only if (f(u'),f(v')) ϵ E.



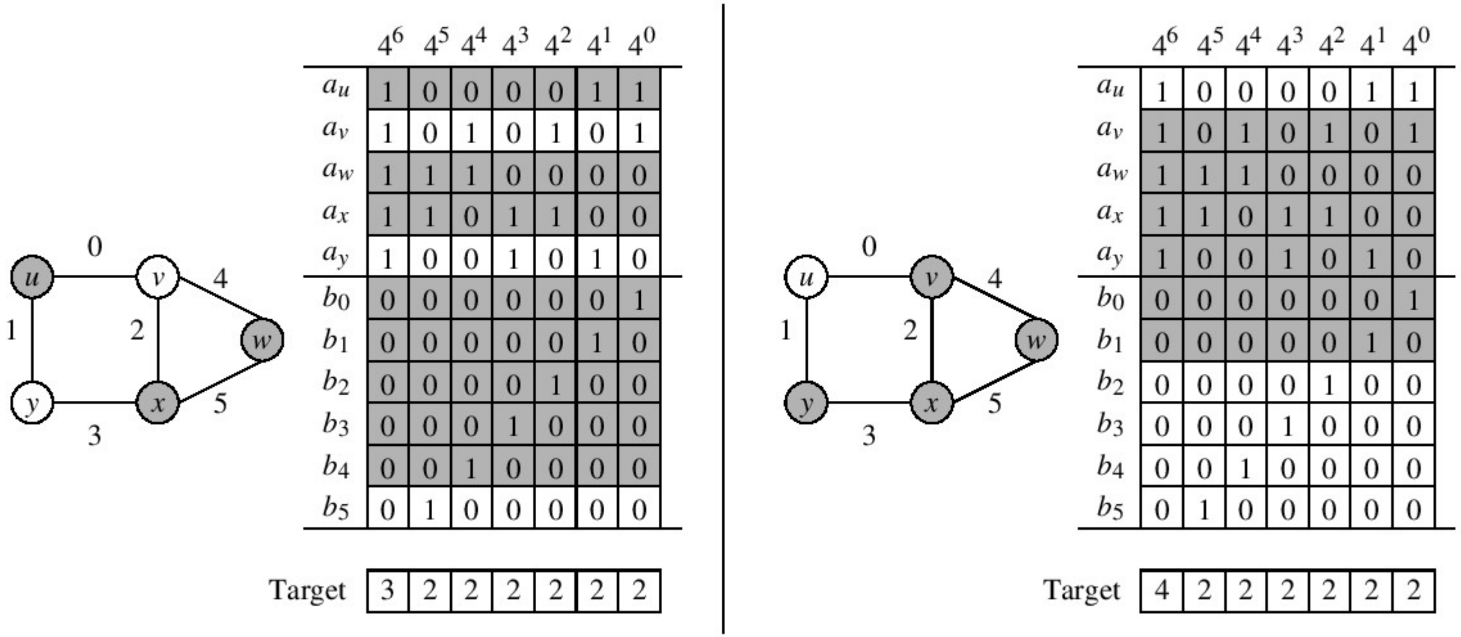
First, I show that SUBGRAPH-ISOMORPHISM is in NP. For an input (G,H) of SUBGRAPH-ISOMORPHISM, guess a subset $V' \subseteq V(G)$ together with a bijective function $f : V' \to V(H)$. Then, check whether f preserves the adjacency relations of V' in $V(H)$.

In order to show the NP-Hardness, I reduce CLIQUE to SUBGRAPH-ISOMORPHISM. Let (G,r) be an input for CLIQUE. We construct the input (G,K_r) for SUBGRAPH-ISOMORPHISM, where K_r is the complete graph on r vertices. G has an r-clique if and only if G has a subgraph isomorphic to K_r. This reduction evidently runs in polynomial time.

3. (a) Prove that the SUBSET-SUM problem is NP-Complete.

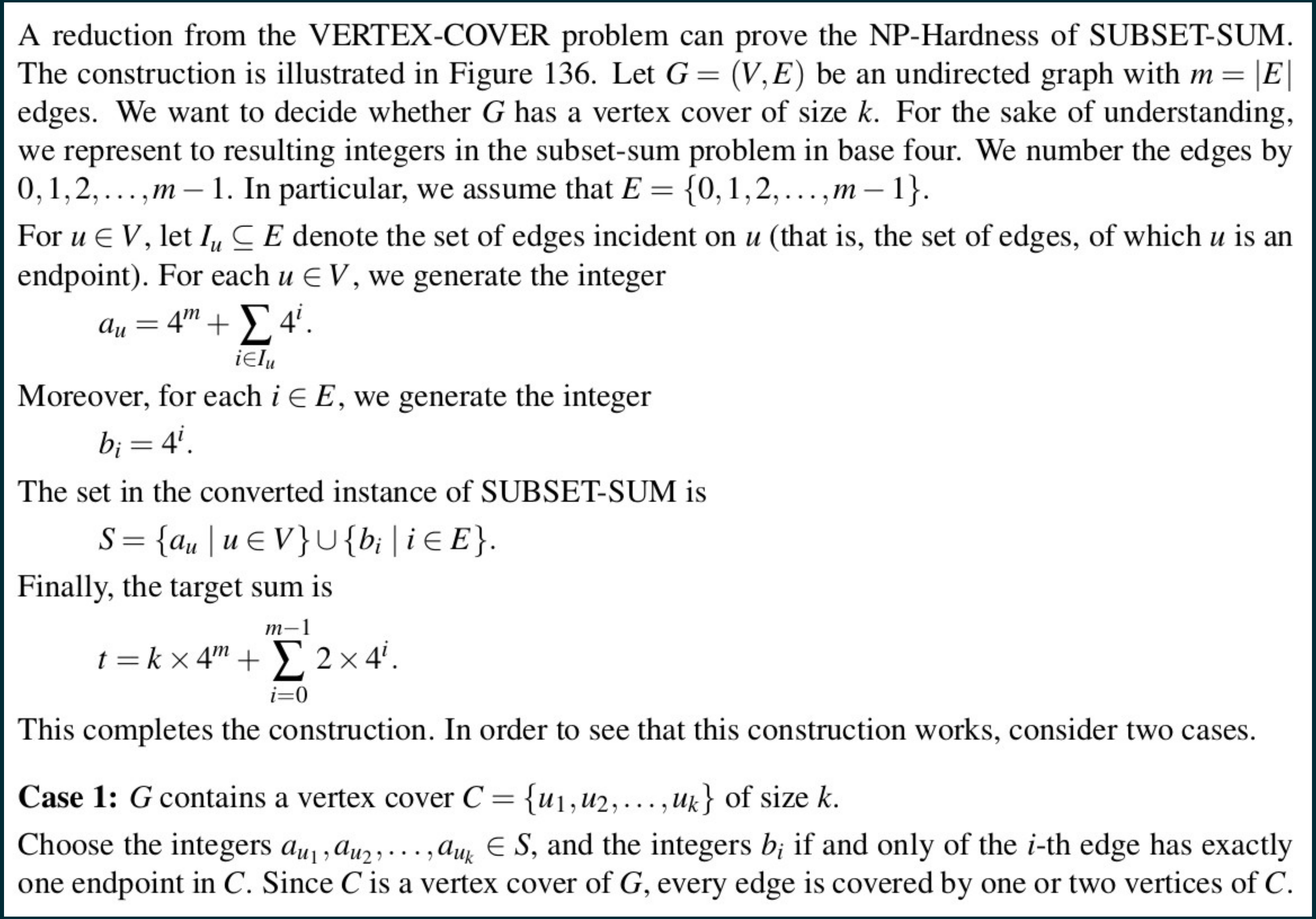
Given: n positive integers a_1, a_2, $\cdots$, a_n, and a target sum t. Decide whether there is a subcollection of the given integers that add up to t.

Figure 136: Reduction from VERTEX-COVER to SUBSET-SUM



	4^6	4^5	4^4	4^3	4^2	4^1	4^0
a_u	1	0	0	0	0	1	1
a_v	1	0	1	0	1	0	1
a_w	1	1	1	0	0	0	0
a_x	1	1	0	1	1	0	0
a_y	1	0	0	1	0	1	0
b_0	0	0	0	0	0	0	1
b_1	0	0	0	0	0	1	0
b_2	0	0	0	0	1	0	0
b_3	0	0	0	1	0	0	0
b_4	0	0	1	0	0	0	0
b_5	0	1	0	0	0	0	0
Target	3	2	2	2	2	2	2

	4^6	4^5	4^4	4^3	4^2	4^1	4^0
a_u	1	0	0	0	0	1	1
a_v	1	0	1	0	1	0	1
a_w	1	1	1	0	0	0	0
a_x	1	1	0	1	1	0	0
a_y	1	0	0	1	0	1	0
b_0	0	0	0	0	0	0	1
b_1	0	0	0	0	0	1	0
b_2	0	0	0	0	1	0	0
b_3	0	0	0	1	0	0	0
b_4	0	0	1	0	0	0	0
b_5	0	1	0	0	0	0	0
Target	4	2	2	2	2	2	2



A reduction from the VERTEX-COVER problem can prove the NP-Hardness of SUBSET-SUM. The construction is illustrated in Figure 136. Let $G = (V, E)$ be an undirected graph with $m = |E|$ edges. We want to decide whether G has a vertex cover of size k. For the sake of understanding, we represent to resulting integers in the subset-sum problem in base four. We number the edges by $0, 1, 2, \ldots, m-1$. In particular, we assume that $E = \{0, 1, 2, \ldots, m-1\}$.

For $u \in V$, let $I_u \subseteq E$ denote the set of edges incident on u (that is, the set of edges, of which u is an endpoint). For each $u \in V$, we generate the integer

$$a_u = 4^m + \sum_{i \in I_u} 4^i.$$

Moreover, for each $i \in E$, we generate the integer

$$b_i = 4^i.$$

The set in the converted instance of SUBSET-SUM is

$$S = \{a_u \mid u \in V\} \cup \{b_i \mid i \in E\}.$$

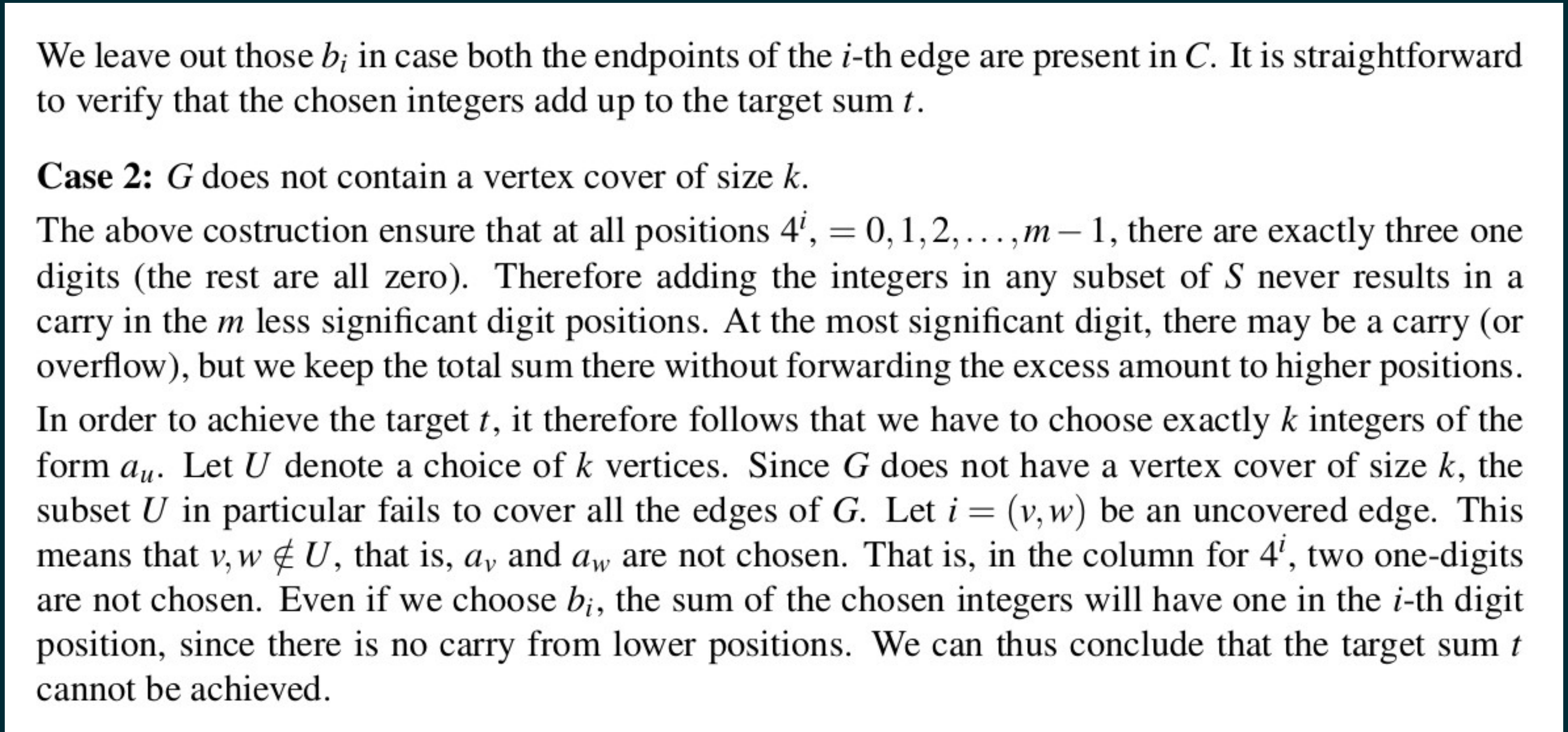
Finally, the target sum is

$$t = k \times 4^m + \sum_{i=0}^{m-1} 2 \times 4^i.$$

This completes the construction. In order to see that this construction works, consider two cases.

Case 1: G contains a vertex cover $C = \{u_1, u_2, \ldots, u_k\}$ of size k.

Choose the integers $a_{u_1}, a_{u_2}, \ldots, a_{u_k} \in S$, and the integers b_i if and only of the i-th edge has exactly one endpoint in C. Since C is a vertex cover of G, every edge is covered by one or two vertices of C.



We leave out those b_i in case both the endpoints of the i-th edge are present in C. It is straightforward to verify that the chosen integers add up to the target sum t.

Case 2: G does not contain a vertex cover of size k.

The above costruction ensure that at all positions 4^i, $= 0,1,2,\ldots,m-1$, there are exactly three one digits (the rest are all zero). Therefore adding the integers in any subset of S never results in a carry in the m less significant digit positions. At the most significant digit, there may be a carry (or overflow), but we keep the total sum there without forwarding the excess amount to higher positions.

In order to achieve the target t, it therefore follows that we have to choose exactly k integers of the form a_u. Let U denote a choice of k vertices. Since G does not have a vertex cover of size k, the subset U in particular fails to cover all the edges of G. Let $i = (v,w)$ be an uncovered edge. This means that $v,w \notin U$, that is, a_v and a_w are not chosen. That is, in the column for 4^i, two one-digits are not chosen. Even if we choose b_i, the sum of the chosen integers will have one in the i-th digit position, since there is no carry from lower positions. We can thus conclude that the target sum t cannot be achieved.

(b) What if we allow the integers to be negative?

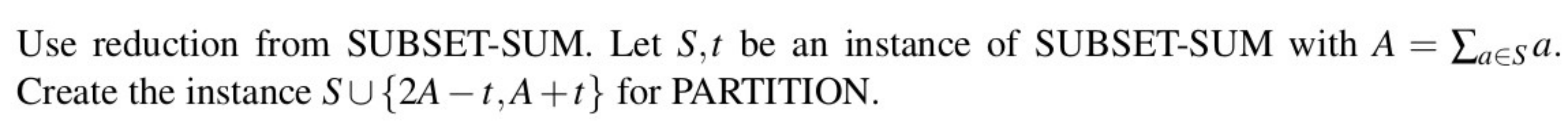
Yes. Any instance of SUBSET-SUM is also an instance of the generalized subset-sum problem.

4. Prove that the PARTITION problem is NP-Complete.

You are given n positive integers a_1, a_2, ⋯, a_n such that

a_1 + a_2 + ⋯ + a_n = A is even.

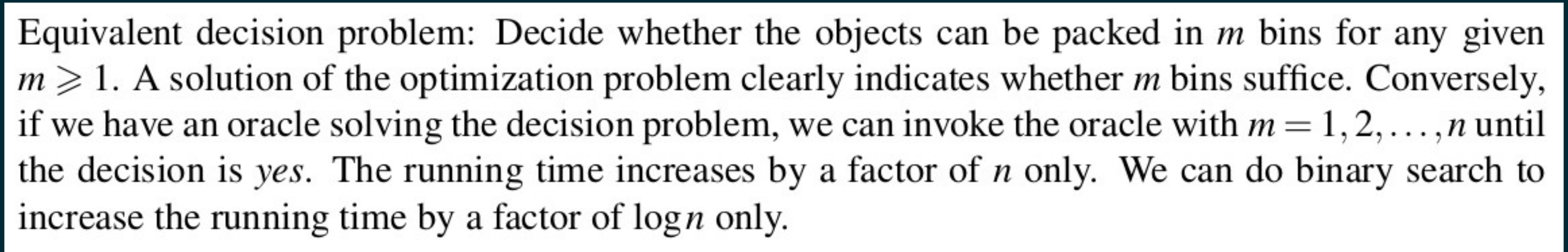
Decide whether the given integers can be partitioned into two subcollections such that the sum of the integers in each subcollection is A / 2.



Use reduction from SUBSET-SUM. Let S,t be an instance of SUBSET-SUM with $A = \sum_{a \in S} a$. Create the instance $S \cup \{2A - t, A + t\}$ for PARTITION.

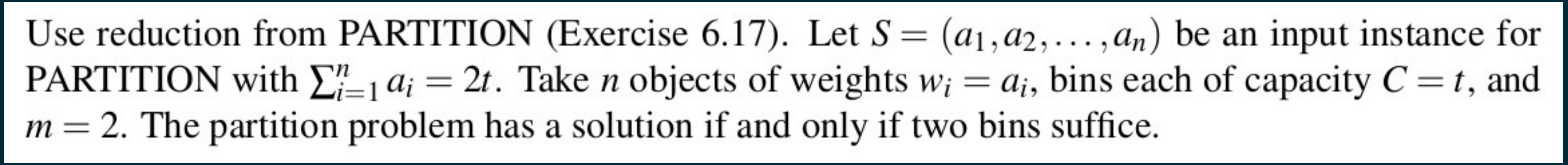
5. BIN-PACKING problem: You are given n objects of weights w_1, w_2, ···, w_n. You are given an infinite supply of bins each with weight capacity C. You want to pack all the objects in m bins such that m is as small as possible. (Assume that each w_i ≤ C.)

(a) Frame an equivalent decision problem, and prove that the decision problem can be solved in poly-time if and only if the optimization problem can be solved in poly-time.



Equivalent decision problem: Decide whether the objects can be packed in m bins for any given $m \geqslant 1$. A solution of the optimization problem clearly indicates whether m bins suffice. Conversely, if we have an oracle solving the decision problem, we can invoke the oracle with $m = 1, 2, \ldots, n$ until the decision is *yes*. The running time increases by a factor of n only. We can do binary search to increase the running time by a factor of $\log n$ only.

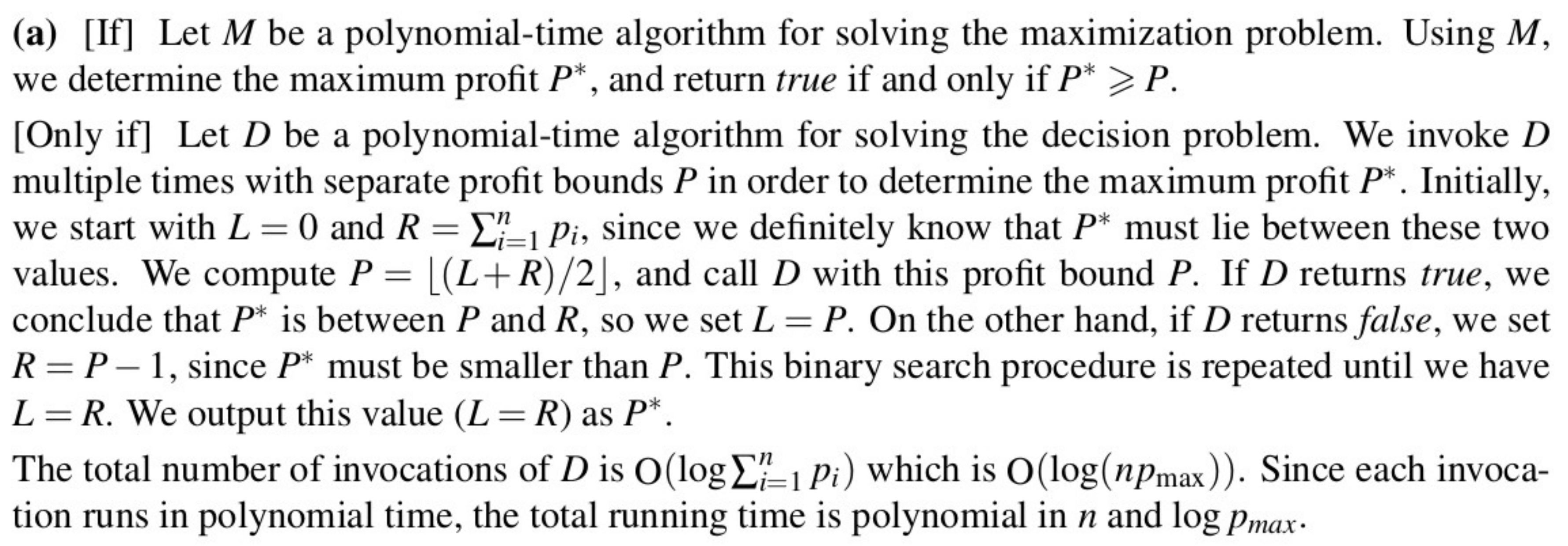
(b) Prove that the decision version is NP-Complete.



Use reduction from PARTITION (Exercise 6.17). Let $S = (a_1, a_2, \ldots, a_n)$ be an input instance for PARTITION with $\sum_{i=1}^{n} a_i = 2t$. Take n objects of weights $w_i = a_i$, bins each of capacity $C = t$, and $m = 2$. The partition problem has a solution if and only if two bins suffice.

6. KNAPSACK problem: n objects of weights w_1, w_2, ⋯, w_n and integral profits p_1, p_2, ⋯, p_n. There is a knapsack of capacity C. Pack objects in the knapsack without exceeding the capacity. The goal is to maximize the profit of packed objects.

(a) Formulate an equivalent decision problem.

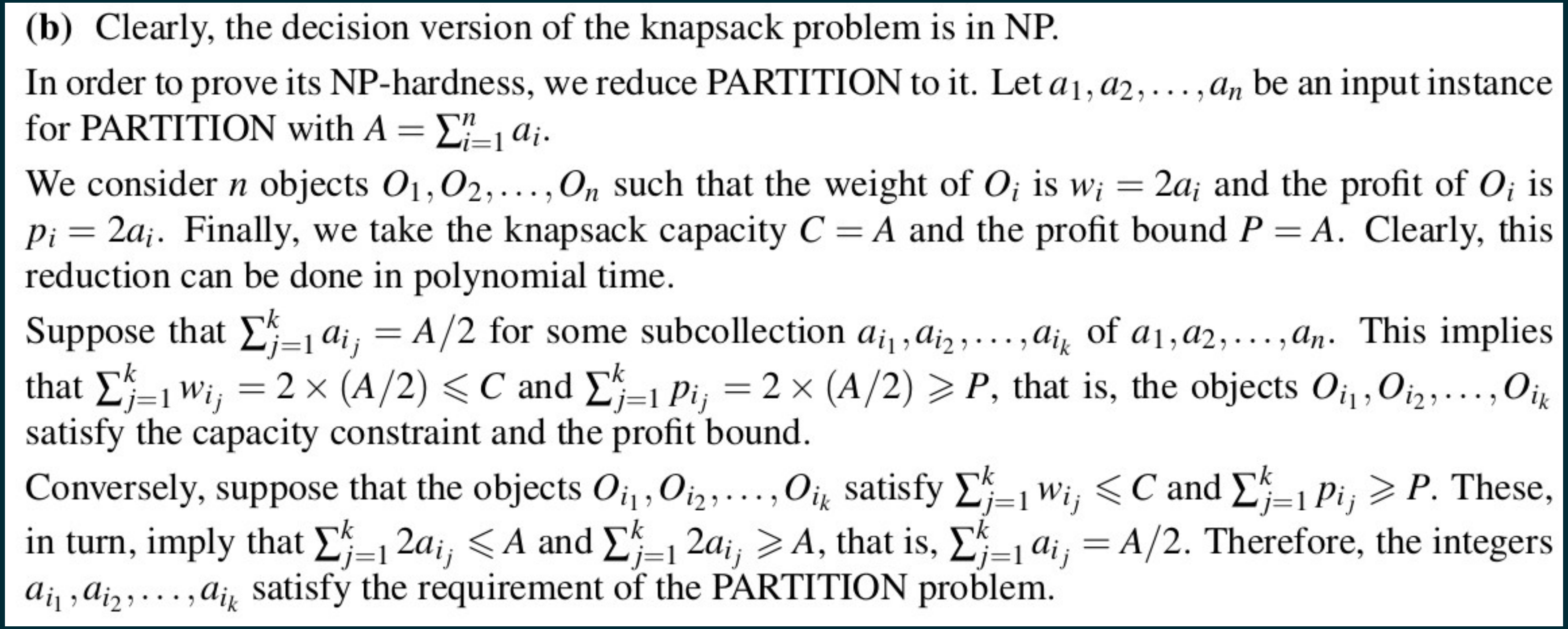


(a) [If] Let M be a polynomial-time algorithm for solving the maximization problem. Using M, we determine the maximum profit P^*, and return *true* if and only if $P^* \geqslant P$.

[Only if] Let D be a polynomial-time algorithm for solving the decision problem. We invoke D multiple times with separate profit bounds P in order to determine the maximum profit P^*. Initially, we start with $L = 0$ and $R = \sum_{i=1}^{n} p_i$, since we definitely know that P^* must lie between these two values. We compute $P = \lfloor (L+R)/2 \rfloor$, and call D with this profit bound P. If D returns *true*, we conclude that P^* is between P and R, so we set $L = P$. On the other hand, if D returns *false*, we set $R = P - 1$, since P^* must be smaller than P. This binary search procedure is repeated until we have $L = R$. We output this value ($L = R$) as P^*.

The total number of invocations of D is $O(\log \sum_{i=1}^{n} p_i)$ which is $O(\log(np_{\max}))$. Since each invocation runs in polynomial time, the total running time is polynomial in n and $\log p_{max}$.

(b) Prove that the decision problem is NP-Complete.



(b) Clearly, the decision version of the knapsack problem is in NP.

In order to prove its NP-hardness, we reduce PARTITION to it. Let $a_1, a_2, \ldots, a_n$ be an input instance for PARTITION with $A = \sum_{i=1}^{n} a_i$.

We consider n objects $O_1, O_2, \ldots, O_n$ such that the weight of O_i is $w_i = 2a_i$ and the profit of O_i is $p_i = 2a_i$. Finally, we take the knapsack capacity $C = A$ and the profit bound $P = A$. Clearly, this reduction can be done in polynomial time.

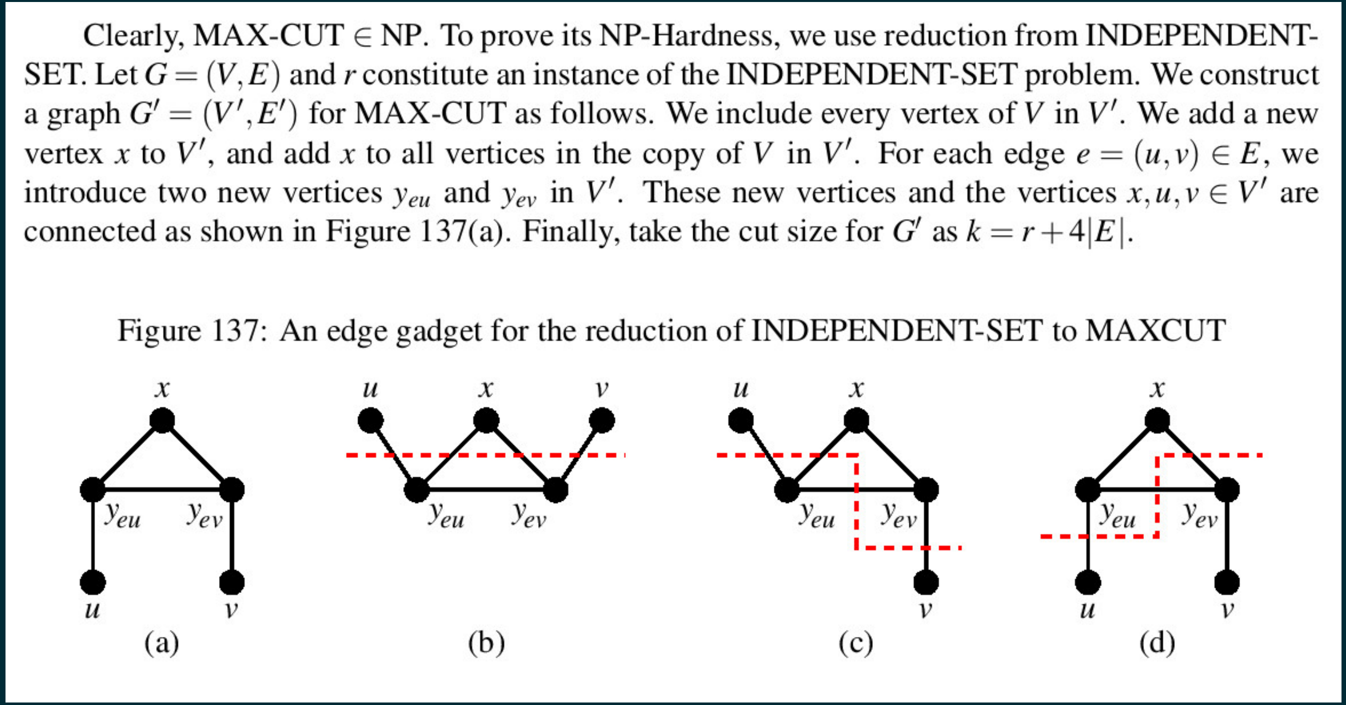
Suppose that $\sum_{j=1}^{k} a_{i_j} = A/2$ for some subcollection $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ of $a_1, a_2, \ldots, a_n$. This implies that $\sum_{j=1}^{k} w_{i_j} = 2 \times (A/2) \leqslant C$ and $\sum_{j=1}^{k} p_{i_j} = 2 \times (A/2) \geqslant P$, that is, the objects $O_{i_1}, O_{i_2}, \ldots, O_{i_k}$ satisfy the capacity constraint and the profit bound.

Conversely, suppose that the objects $O_{i_1}, O_{i_2}, \ldots, O_{i_k}$ satisfy $\sum_{j=1}^{k} w_{i_j} \leqslant C$ and $\sum_{j=1}^{k} p_{i_j} \geqslant P$. These, in turn, imply that $\sum_{j=1}^{k} 2a_{i_j} \leqslant A$ and $\sum_{j=1}^{k} 2a_{i_j} \geqslant A$, that is, $\sum_{j=1}^{k} a_{i_j} = A/2$. Therefore, the integers $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ satisfy the requirement of the PARTITION problem.

7. MAX-CUT: Let G = (V,E) be an undirected graph. Partition V into two subsets X and Y such the number of edges connecting X and Y is maximized.

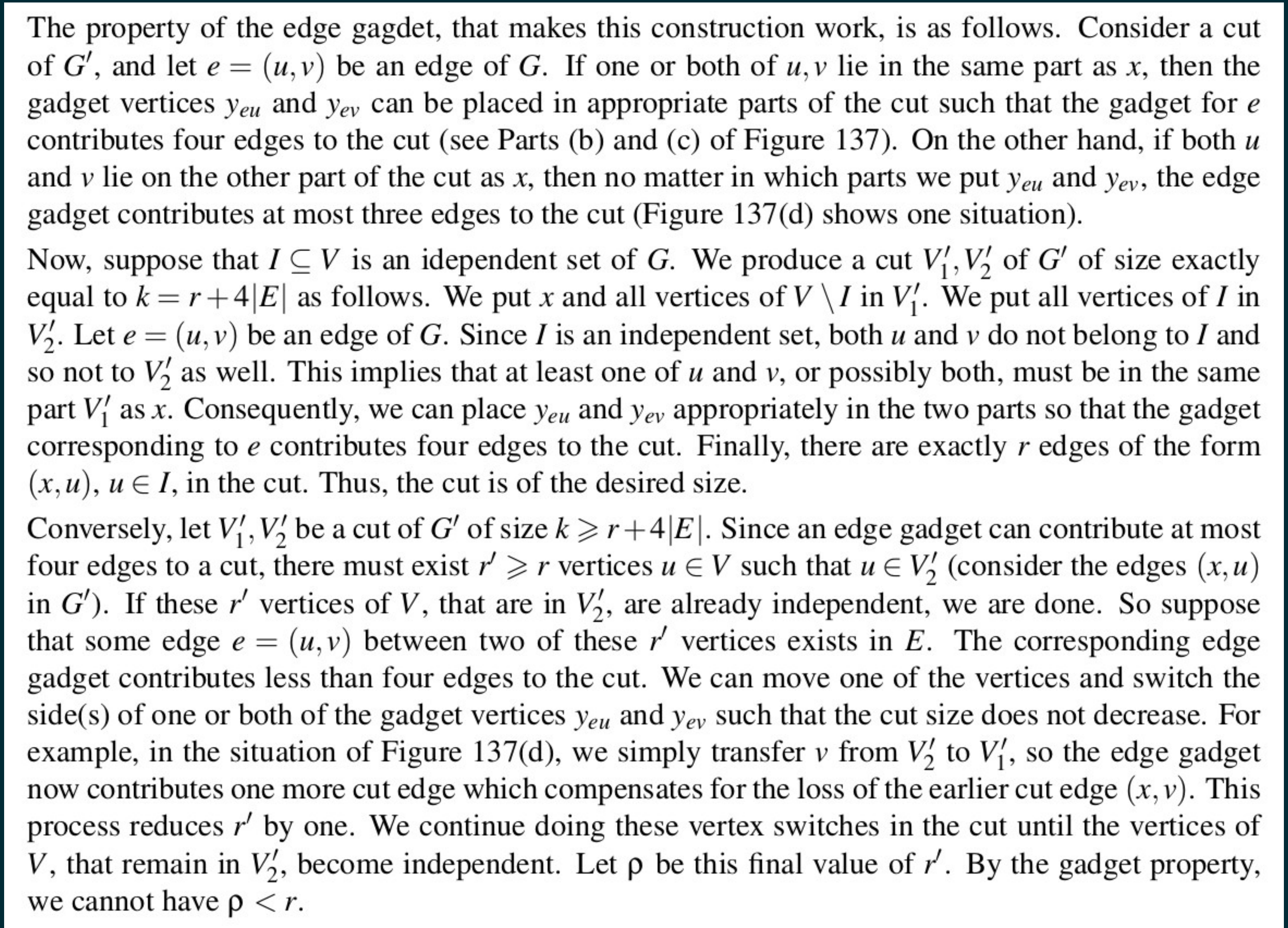
(a) Supply an equivalent decision problem.

(b) Prove that the decision problem is NP-Complete.



Clearly, MAX-CUT $\in$ NP. To prove its NP-Hardness, we use reduction from INDEPENDENT-SET. Let $G = (V,E)$ and r constitute an instance of the INDEPENDENT-SET problem. We construct a graph $G' = (V',E')$ for MAX-CUT as follows. We include every vertex of V in V'. We add a new vertex x to V', and add x to all vertices in the copy of V in V'. For each edge $e = (u,v) \in E$, we introduce two new vertices y_{eu} and y_{ev} in V'. These new vertices and the vertices $x,u,v \in V'$ are connected as shown in Figure 137(a). Finally, take the cut size for G' as $k = r + 4|E|$.

Figure 137: An edge gadget for the reduction of INDEPENDENT-SET to MAXCUT



The property of the edge gagdet, that makes this construction work, is as follows. Consider a cut of G', and let $e = (u, v)$ be an edge of G. If one or both of u, v lie in the same part as x, then the gadget vertices y_{eu} and y_{ev} can be placed in appropriate parts of the cut such that the gadget for e contributes four edges to the cut (see Parts (b) and (c) of Figure 137). On the other hand, if both u and v lie on the other part of the cut as x, then no matter in which parts we put y_{eu} and y_{ev}, the edge gadget contributes at most three edges to the cut (Figure 137(d) shows one situation).

Now, suppose that $I \subseteq V$ is an idependent set of G. We produce a cut V_1', V_2' of G' of size exactly equal to $k = r + 4|E|$ as follows. We put x and all vertices of $V \setminus I$ in V_1'. We put all vertices of I in V_2'. Let $e = (u, v)$ be an edge of G. Since I is an independent set, both u and v do not belong to I and so not to V_2' as well. This implies that at least one of u and v, or possibly both, must be in the same part V_1' as x. Consequently, we can place y_{eu} and y_{ev} appropriately in the two parts so that the gadget corresponding to e contributes four edges to the cut. Finally, there are exactly r edges of the form (x, u), $u \in I$, in the cut. Thus, the cut is of the desired size.

Conversely, let V_1', V_2' be a cut of G' of size $k \geqslant r + 4|E|$. Since an edge gadget can contribute at most four edges to a cut, there must exist $r' \geqslant r$ vertices $u \in V$ such that $u \in V_2'$ (consider the edges (x, u) in G'). If these r' vertices of V, that are in V_2', are already independent, we are done. So suppose that some edge $e = (u, v)$ between two of these r' vertices exists in E. The corresponding edge gadget contributes less than four edges to the cut. We can move one of the vertices and switch the side(s) of one or both of the gadget vertices y_{eu} and y_{ev} such that the cut size does not decrease. For example, in the situation of Figure 137(d), we simply transfer v from V_2' to V_1', so the edge gadget now contributes one more cut edge which compensates for the loss of the earlier cut edge (x, v). This process reduces r' by one. We continue doing these vertex switches in the cut until the vertices of V, that remain in V_2', become independent. Let ρ be this final value of r'. By the gadget property, we cannot have $\rho < r$.