Lecture 30
ECE 664

NP-Completeness Proofs for 3DM, VC, CLIQUE, HC, and PARTITION

AVI KAK
PURDUE U.

3DM: Read the explanation that follows the statement of this problem in the Lecture 29 scroll. The NP-Completeness of 3DM is established through the polynomial transformation:

$$\text{3DM} \xrightarrow{x} \text{3SAT}$$

In the discussion below, i always goes from 1 to n and j from 1 to m. $n = |U|$ and $m = |C|$.

Let an arbitrary instance of 3SAT involve the Boolean variables $U = \{u_1, u_2, \ldots, u_n\}$ and the 3-literal clauses $C = \{c_1, c_2, \ldots, c_m\}$. Let the truth assignment to U for this instance be given by $t: U \rightarrow \{\text{True}, \text{False}\}$. Now we must construct a mapped instance of 3DM for the given instance of 3SAT. We obviously need to specify the sets W, X, and Y for the 3DM instance. These three sets must be constructed in such a way that they capture both the 3-literal clauses of the 3SAT instance and the truthvalue assignment in that instance. We would also need to specify a set $M \subseteq W \times X \times Y$ in such a way that the truthvalue assignment in the 3SAT instance is satisfying iff M contains a matching. The objects in the set W will look like $u_i[j]$ and $\bar{u}_i[j]$. For now you may think that if c_j includes the literal u_i, we place the object $u_i[j]$ in W, and if c_j includes the literal $\bar{u}_i$, we place the object $\bar{u}_i[j]$ in W. (Later we will see that W includes $u_i[j]$ and $\bar{u}_i[j]$ for all i and all j.) The objects in X will look like $s_1[j]$, $a_i[j]$, and $g_1[k]$, and the objects in Y will look like $s_2[j]$, $b_i[j]$ and $g_2[k]$. In all of these objects, the index i refers to the boolean variable u_i and the index j to the jth clause c_j of the 3SAT instance. (The exact role played by these six object types becomes clear after you see how each ~~literal~~ 3-literal clause and the truth assignment of the 3SAT instance is translated into the triples for the set M of the mapped 3DM instance.) We will use the notation c_j^{3DM} to denote the triples for M as generated by the clause c_j of the 3SAT instance:

$$\boxed{c_j^{3DM} = \left\{ \left(u_i[j], s_1[j], s_2[j]\right) \,\middle|\, u_i \in c_j \right\} \cup \left\{ \left(\bar{u}_i[j], s_1[j], s_2[j]\right) \,\middle|\, \bar{u}_i \in c_j \right\}}$$

Consider the example $U = \{u_1, u_2, u_3\}$ and $C = \{\{u_1, \bar{u}_2, u_3\}, \{\bar{u}_1, \bar{u}_2, u_3\}\}$. In this case

$$c_1^{3DM} = \left\{ \left(u_1[1], s_1[1], s_2[1]\right), \left(\bar{u}_2[1], s_1[1], s_2[1]\right), \left(u_3[1], s_1[1], s_2[1]\right) \right\}$$

$$c_2^{3DM} = \left\{ \left(\bar{u}_1[2], s_1[2], s_2[2]\right), \left(\bar{u}_2[2], s_1[2], s_2[2]\right), \left(u_3[2], s_1[2], s_2[2]\right) \right\}$$

This construction of the triples for the mapped 3DM instance should make clear the role played by the objects $s_1[j]$ and $s_2[j]$. Given the definition of a matching in 3DM, any one object appears in exactly one triple in a matching. So should we choose the triple $(u_1[1], s_1[1], s_2[1])$ for a matching subset of M, that would preclude our selecting the other two triples from c_1^{3DM} for the matching subset. This amounts to saying that, in the 3SAT instance, we want to try for c_1 to be true by setting u_1 to be true.

- Let's now focus on how we would represent in the mapped 3DM instance a given truth assignment to the variables in U. If a variable u_i is set to "True", we place in M the following triples:

$$T_i^t = \left\{ \left(\bar{u}_i[j], a_i[j], b_i[j]\right) \,\middle|\, 1 \leq j \leq m \right\}$$

m is the number of clauses in 3SAT instance

and if the variable u_i is set to "False", we place in M the following m triples:

$$T_i^f = \left\{ \left(u_i[j], a_i[j+1], b_i[j]\right) \,\middle|\, 1 \leq j < m \right\} \cup \left\{ \left(u_i[m], a_i[1], b_i[m]\right) \right\}$$

Going back to the previous example, if the truth assignment in the 3SAT instance calls for u_1 to be true, we place in M the triples $(\bar{u}_1[1], a_1[1], b_1[1])$ and $(\bar{u}_1[2], a_1[2], b_1[2])$. A consequence of placing these two triples in M is that now we will be forced to select for the matching

set those triples whose first element is $u_i[1]$ or $u_i[2]$ from all the triples placed in M through c_1^{3DM} and c_2^{3DM}. If we choose a triple whose first element is $u_i[1]$ that amounts to saying that we want the first clause to be true because we have set its literal u_i to be true.

- As you surely noticed, both $a_i[\,]$ and $b_i[\,]$ elements in the T_i^t triples carry the same index j, but that is NOT the case with the T_i^f triples. WHY? Including $b_i[j]$ in the triples for both T_i^t and T_i^f makes it impossible to set u_i simultaneously to True and False in the underlying 3SAT instance. And using the staggered j index for T_i^f ensures that either we will select <u>all</u> of the triples T_i^t <u>or</u> <u>all</u> of the triples of T_i^f. In other words, we will not be allowed to intermix some of the triples from T_i^t with the rest of the triples from T_i^f.
- So, are we done? Unfortunately not. To see as to why let's examine the set W. As indicated earlier, we have $u_i[j] \in W$ and $\bar{u}_i[j] \in W$ for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$ where n is the number of Boolean variables in U and m the number of clauses in C. $\boxed{\text{So } |W| = 2nm}$. Therefore, any matching (in the 3DM sense) in the mapped 3DM instance must contain exactly $2mn$ triples. But the truth assignment over all the n variables will give us only mn triples for the matching, and the c_j^{3DM} sets will yield only a total of m triples (one from each c_j^{3DM}) for the matching. So we have a shortfall of $\boxed{2mn - mn - m = m(n-1)}$ with regard to the number of triples needed for a matching. This shortfall is a consequence of the fact that both T_i^t and T_i^f inject the objects $u_i[j]$ and $\bar{u}_i[j]$ that may <u>not</u> correspond to any literals in any of the clauses in the 3SAT instance. So we need additional triples — you can think of these as filler triples — to enable us to construct a 3DM matching. Here are the additional triples:

$$G = \left\{ \left(u_i[j],\, g_1[k],\, g_2[k]\right),\ \left(\bar{u}_i[j],\, g_1[k],\, g_2[k]\right) \;\middle|\; \begin{array}{l} 1 \le k \le m(n-1) \\ 1 \le i \le n \\ 1 \le j \le m \end{array} \right\}$$

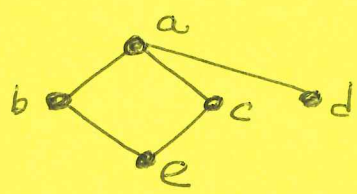
From this G we include in the matching subset only those triples whose first elements do not correspond to the literals in any of the clauses.

- With all of triples defined as above, you should be able to show that the 3SAT instance can be satisfied iff the mapped 3DM instance contains a matching.

VC and CLIQUE:

Despite the fact that VERTEX COVER (VC) and CLIQUE are independently useful for proving NP-Completeness results, they are really just two different ways of looking at the same problem. To see that, it is convenient to consider them in conjunction with a third problem called the INDEPENDENT SET (INDSET) problem.

- An independent set in a graph $G = (V, E)$ is a ~~subset~~ subset $V' \subseteq V$ such that for all $u, v \in V'$, the edge $\{u, v\}$ is <u>not</u> in E.



- Example: Consider the graph on the right. The following edges $\{b,c\}$, $\{a,e\}$, $\{b,d\}$, $\{c,d\}$ and $\{e,d\}$ are <u>not</u> in E. But we cannot lump together all the vertices associated with these non-existant edges to construct an independent set. To construct an independent set, we could start with vertex 'a' and then consider each ~~of~~ of the other vertices, one at a time, to see if they can be in an independent set along with 'a'. Reasoning in this manner, we end up with the set $\{a, e\}$ as an independent set. However, if we start with the vertex 'b', we end up with $\{b, c, d\}$ as an independent set.

We are now ready to pose the INDEPENDENT SET decision problem.

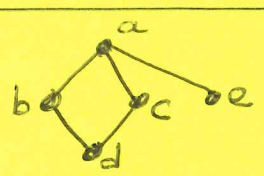
INDEPENDENT SET
INSTANCE : A graph $G=(V,E)$ and a positive integer $J \leq |V|$
QUESTION : Does G contain an independent set V' with $|V'| \geq J$?

- The following lemma states the relationship between independent set, vertex cover, and clique :

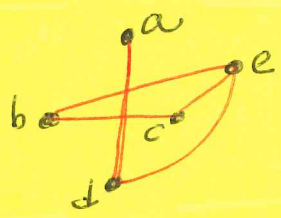
Lemma : For any graph $G=(V,E)$ and subset $V' \subseteq V$, the following statements are equivalent :

(a) V' is a vertex cover for G
(b) $V-V'$ is an independent set for G
(c) $V-V'$ is a clique in the complement G^c of G where $G^c = (V, E^c)$ with $E^c = \{\{u,v\} \mid u,v \in V \text{ and } \{u,v\} \notin E\}$

- To illustrate the lemma, consider the graph shown at the right. One possible vertex cover for this graph is $\{a,d\}$. ~~Yet~~ Another vertex cover is $\{b,c,e\}$. Yet another vertex cover here is $\{b,c,a\}$.



Consider the vertex cover $\{a,d\}$. That is $V'=\{a,d\}$. Then $V-V' = \{b,c,e\}$. It is evident that $\{b,c,e\}$ is an independent set since no two vertices in $\{b,c,e\}$ are connected. Let's now examine G^c:
It is clear that $V-V' = \{b,c,e\}$ constitutes a clique in G^c since all the nodes in $\{b,c,e\}$ are connected to one another.



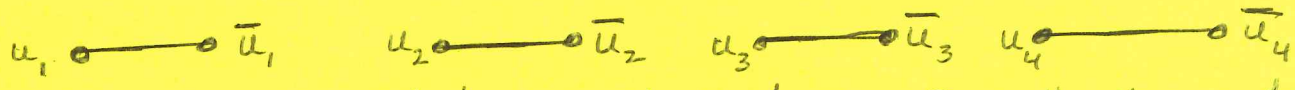
- The above lemma demonstrates that the three problems — VC, INDSET, and CLIQUE — can be regarded as different versions of one another. The lemma also points to how we can transform any one of the three problems into either of the other two. For example, to transform VC to CLIQUE, let $G=(V,E)$ and $K \leq |V|$ constitute an instance of VC, the corresponding instance of CLIQUE is the graph G^c and the integer $J = |V| - K$.
- We will now review the salient arguments of the NP-Completeness proof for VC :

THEOREM : VERTEX COVER (VC) is NP-Complete.

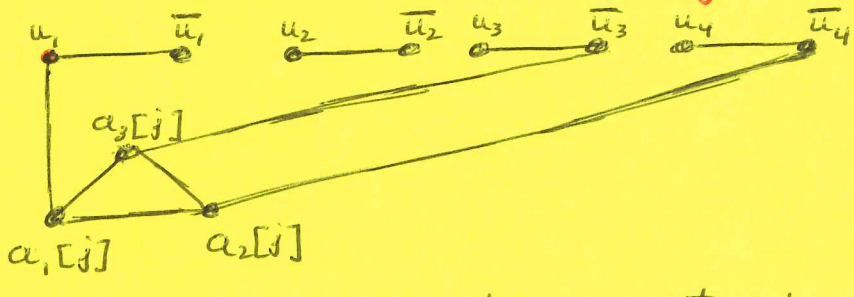
PROOF : It is trivially established that $VC \in NP$. For the rest of the proof we will construct the polynomial transformation :

$$\boxed{VC \xrightarrow{\propto} 3SAT}$$

Let $U = \{u_1, u_2, \ldots, u_n\}$ and $C = \{c_1, c_2, \ldots, c_m\}$ with each c_i being a 3-literal clause be an instance of 3SAT. We must now construct a graph $G=(V,E)$ and specify a value for the positive integer $K \leq |V|$ such that G has a vertex cover of size K or less iff C is satisfiable. To construct such a graph G from the 3SAT instance, we represent each $u_i \in U$ by an edge with vertex labels u_i and $\bar{u}_i$. So if $U = \{u_1, u_2, u_3, u_4\}$, we will end up with the following edges :

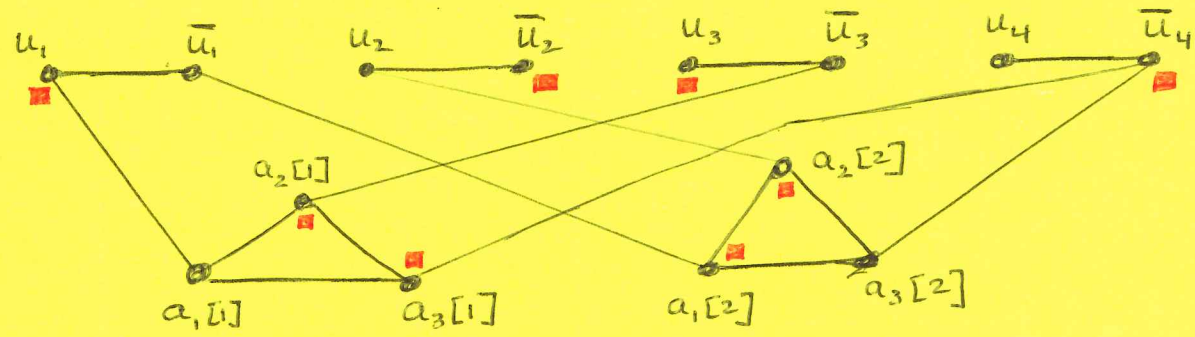


Each clause will be represented by a triangle in G, with the vertex labels $a_1[j]$, $a_2[j]$, and $a_3[j]$ for the three vertices of the triangle corresponding to the jth clause. The exact literals used in the c_j clause will be indicated in the graph by connecting the triangle vertices with the vertices shown above. For example, if $c_j = \{u_1, \bar{u}_3, \bar{u}_4\}$, we obtain the construction :



With G constructed in this manner, we next specify $\boxed{K = n + 2m}$. Note that the smallest vertex cover for G will be of size $n+2m$ since each of n edges that correspond to the n boolean variables must contribute at least one vertex to the vertex cover and since each ~~triangle~~ of the m triangles must contribute at least 2 vertices to

the same. In this construction, the vertices of the triangles that are NOT included in the vertex cover will correspond to the satisfying truth assignment for the 3SAT instance. To illustrate, consider the 3SAT instance $U = \{u_1, u_2, u_3, u_4\}$ and $C = \{\{u_1, \bar{u}_3, \bar{u}_4\}, \{\bar{u}_1, u_2, \bar{u}_4\}\}$. This gives rise to G:



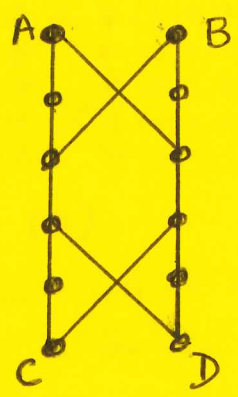
■ : Vertex cover vertices

since $a_1[1]$ is connected to u_1 and NOT to $\bar{u}_1$

In the first triangle, the first vertex, $a_1[1]$, is not included in the vertex cover, so $\boxed{t(u_1) = \text{True}}$ in the satisfying truth assignment for the 3SAT instance. In the second triangle, the third vertex, $a_3[2]$, is not in the vertex cover. Since $a_3[2]$ is connected to $\bar{u}_4$; we set $\boxed{t(u_4) = \text{False}}$ in the satisfying truth assignment. For the other two variables, u_2 and u_3, we can be arbitrary in choosing their truthvalues.

Exercise: For a 3SAT instance, use $U = \{u_1, u_2, u_3\}$ and let C be the set of eight 3-literal clauses that cannot be satisfied by any truth assignment to U. (You can construct C by listing all possible truth assignments for U and, for each truth assignment, writing down a clause that cannot be satisfied by that truth assignment. C will be the set of all such clauses.) Now construct a VC instance for this instance of 3SAT. You will obviously have to set $K = 3 + 2 \times 8 = 19$. Now show by construction that the graph G in the VC instance will not have a vertex cover with $K = 19$.

HC: We will establish the NP-Completeness of HC by constructing the polynomial transformation VC $\propto$ HC. Let $G = (V, E)$ and a positive integer K represent an arbitrary instance of VC and let $G' = (V', E')$ represent the mapped instance of HC. To construct G' from G, let's first focus on isolated edges in G. Let $e = \{u, v\} \in E$ be such an edge in G. For such an edge in G, we will place in G' the vertices and edges shown at (a) below:



(a)

A = u_connector
C = u_connector
B = v_connector
D = v_connector

As to why we refer to A and C as u-connector nodes, and B and D as v-connector nodes, you will soon find out

In order to see why the above structure is useful in a polynomial transformation from G to G', consider the following YES-instance of VC:

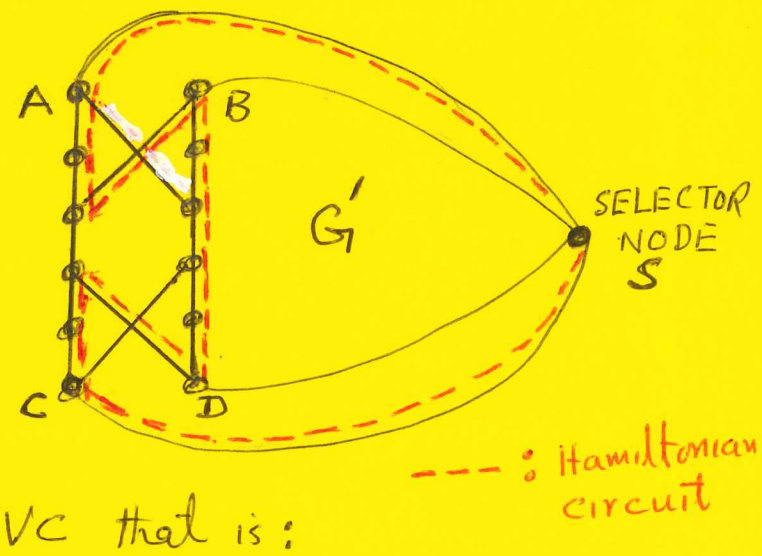
$$\boxed{G = (V, E) \text{ with } V = \{u, v\} \text{ and } E = \{\{u, v\}\} \text{ and } VC = \{u\}}$$

This is a YES-instance of VC with $K = 1$. (See the definition of VC in Lecture 29) Note that G has only two nodes and one edge.

- Polynomial transformation from VC to HC calls for representing each "isolated" edge of G by the structure shown above. In addition, we must specify a set of "selector nodes" — the number of selector nodes must equal the number of nodes in the vertex cover for G. Since we have only one node in the vertex cover in our YES-instance, we need only

one selector node. Finally, we must join each selector node (which, in our example, is just one node) with every connector node. The resulting G' is shown below:

IMPORTANT: Note that the Hamiltonian circuit connects with the 12-node structure at its u-connector nodes A and C. That's because the vertex cover in the YES-instance of VC has the node u in it. The graph G' shown at right is evidently a YES-instance of HC.

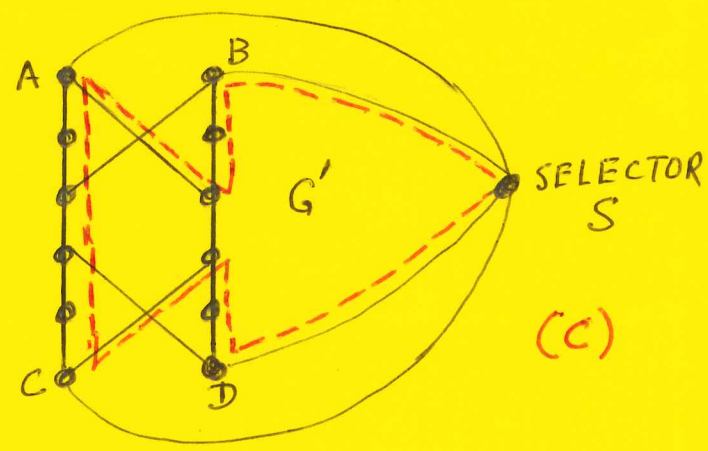


- Let's now consider another YES-instance of VC that is:

$$G = (V, E) \quad V = \{u, v\} \quad E = \{\{u, v\}\} \quad VC = \{v\}$$

The only difference between this YES-instance of VC and the previous YES-instance is that we now have a different vertex cover: $VC = \{v\}$. Since we still have only one vertex in the vertex cover, we will need only one selector node for G'. So it should not be surprising that we end up with the same G' as shown above. However, our Hamiltonian circuit will now be different since the selector will connect with the 12-node structure at the v-connector nodes, as shown in (c) below.

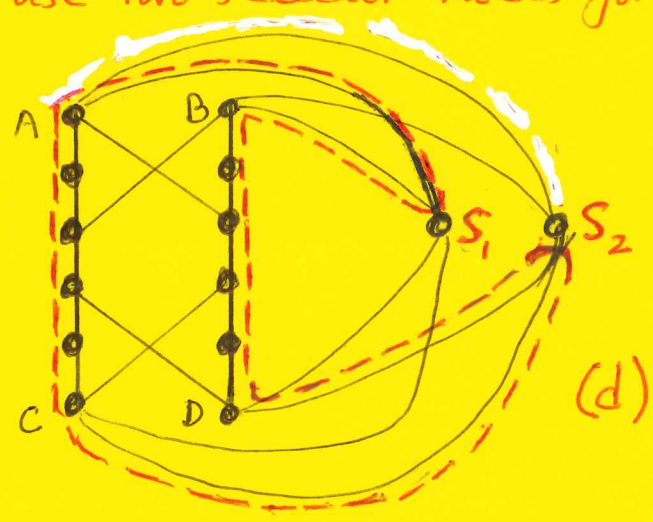
- Now you should be able to see why we refer to A and C as the u-connector nodes and B and D as the v-connector nodes in G' for the edge $\{u, v\}$ in G. The idea is to create a mental association between the node u of G and all six nodes in the left vertical component of the 12-node structure, and the node v of G and all six nodes in the right vertical component of the same structure.



- Continuing with our 1-edge graph for VC, let's now consider the following YES-instance of VC:

$$G = (V, E), \quad V = \{u, v\}, \quad E = \{\{u, v\}\}, \quad VC = \{u, v\}$$

Our graph G remains the same as before. However, our vertex cover now has two nodes in it. This implies we must use two selector nodes for G': The two selector nodes are labeled S_1 and S_2 in the G' graph at right. As you can see, we again have a YES-instance of HC.



- So far we have only considered single edge graphs in our YES-instances of VC. Let's now consider the following YES instance of VC:

$$G = (V, E) \quad V = \{u, v, w\} \quad E = \{\{u, v\}, \{v, w\}\} \quad VC = \{v\}$$

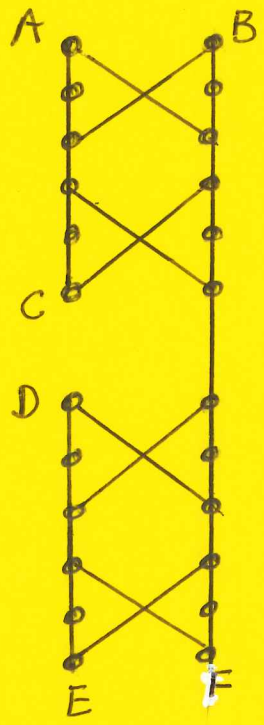
Note that the two edges of G have vertex v in common. We take care of the commonality of a vertex between two edges of G by creating in G' a conjoined structure as shown at the top right on the next page.

In what is shown at right, the upper 12-node structure corresponds to edge $\{u, v\}$ in G, and the lower 12-node structure to the edge $\{v, w\}$ in G. Whereas the vertex u of G is represented by the 6 nodes of G' that go from A to C, the vertex v is represented by the 12 nodes that go all the way from B to F. Vertex w of G is again represented by the 6 nodes on the line from D to E.

A = u-connector
C = u-connector

B = v-connector
F = v-connector

D = w-connector
E = w-connector



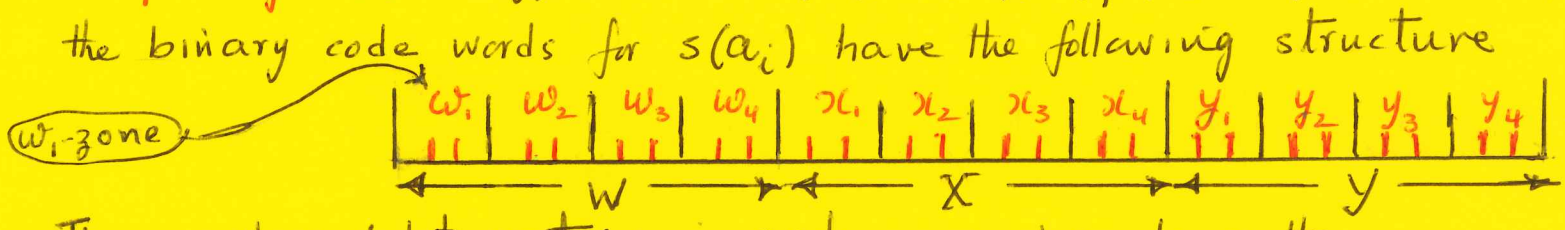
- Since our YES-instance of VC shown at the bottom of the previous page has only one node in the vertex cover, we are allowed only one selector node in G'. We must connect this selector to all 6 connector nodes A, C, B, F, D, and E in G'. Now it is your job to show that you have constructed a YES-instance of HC in G'.
- Your next job is to show that if had started out with a NO-instance of VC, you would NOT be able construct a YES-instance of HC.

PARTITION

PARTITION $\xrightarrow{\propto}$ 3DM

NP-Completeness proof PARTITION is based on a polynomial transformation from 3DM to PARTITION. Let the disjoint sets W, X, Y of equal cardinality and $M \subseteq W \times X \times Y$ constitute a YES-instance of 3DM. That means M contains a matching for W, X, and Y. The goal of the polynomial transformation is to transform a YES-instance of 3DM into a YES-instance of PARTITION in such a way that a NO-instance of 3DM will only go into a NO-instance of PARTITION. Recall that an instance of PARTITION consists of a set $A = \{a_1, a_2, \ldots, a_n\}$ along with a size function $s(a_i)$. The decision question for PARTITION is whether there exists a subset $A' \subset A$ such that $\sum_{a_i \in A'} s(a_i) = \sum_{a_i \in A - A'} s(a_i)$.

- Let $|M| = k$ for the YES-instance of 3DM. In the polynomial transformation from 3DM to PARTITION, we associate an $a_i \in A$ with each triple in M. The size $s(a_i)$ assigned to a_i depends on which elements of W, X, and Y are in the corresponding triple of M. We also place in A two more elements, b_1 and b_2, whose sizes $s(b_1)$ and $s(b_2)$ are such that if M contains a matching then A will possess a partition of the sort we want. $A = \{a_1, a_2, \ldots, a_k, b_1, b_2\}$
- All size values — $s(a_i)$, $s(b_1)$, $s(b_2)$ — consist of binary code words. Each such binary words consists of a bit field for W, followed by a bit field for X, and that is followed by a bit field for Y. Only one bit is set in each bit field for a given a_i depending on which elements of W, X, Y are in the corresponding triple of M. Say, when $W = \{w_1, w_2, w_3, w_4\}$, $X = \{x_1, x_2, x_3, x_4\}$, $Y = \{y_1, y_2, y_3, y_4\}$, the binary code words for $s(a_i)$ have the following structure

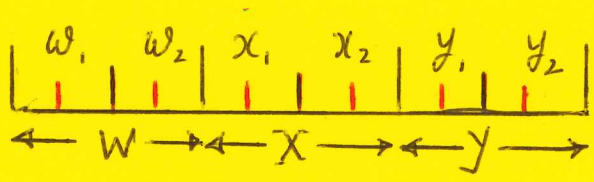


In the bit fields for W, X, and Y we have a "zone" of bit positions reserved for each element of the corresponding set

The number of bit positions in each zone depends on the cardinality of M. Our goal is that when we add all the $s(a_i)$, we do NOT want bits from one zone overflowing into

adjacent zones. Let p denote the number of bit positions reserved for each zone. It is easy to show that $p = \lceil \log_2(k+1) \rceil$ where $k = |M|$: For each $s(a_i)$, only one bit is set in the W bit field, one bit in the X bit field, and one bit in the Y bit field. When calculating $\Sigma s(a_i)$, all the set bits may line up. When 4 bits line up, you need $\lceil \log_2(4+1) \rceil$ bit positions to express the result. When 8 bits line up, you need $\lceil \log_2(8+1) \rceil$ bit positions to express the result. $\lceil x \rceil$ is the smallest integer no less than x.

- To see how the binary code word template shown at the bottom of the previous page can be used for setting $s(a_i)$, let's look at the following YES-instance of 3DM: $W = \{w_1, w_2\}$, $X = \{x_1, x_2\}$, $Y = \{y_1, y_2\}$ and $M = \{(w_1, x_2, y_1), (w_2, x_1, y_2), (w_1, x_1, y_1)\}$. We have $k = |M| = 3$. Therefore, $p = \lceil \log_2(3+1) \rceil = 2$. In this case, the binary code word template looks like



We have a total of 6 zones. Each zone gets 2 bit positions. ($p = 2$)

- In the mapped instance of PARTITION, we have $A = \{a_1, a_2, a_3, b_1, b_2\}$ with a_1 corresponding to the triple (w_1, x_2, y_1), a_2 to the triple (w_2, x_1, y_2), and a_3 to the triple (w_1, x_1, y_1). For each $s(a_i)$, we set the rightmost bit in each zone to 1 for each element of the triple that corresponds to a_i.

$s(a_1)$:	0 1	0 0	0 0	0 1	0 1	0 0
$s(a_2)$:	0 0	0 1	0 1	0 0	0 0	0 1
$s(a_3)$:	0 1	0 0	0 1	0 0	0 1	0 0
$s(a_1) + s(a_2)$:	0 1	0 1	0 1	0 1	0 1	0 1

The triples for a_1 and a_2 constitute a matching. The sum $\Sigma s(a_i)$ for all such triples in M looks like

- We will use B and U to denote the following two different sums:

$$U = \sum_{i=1}^{k} s(a_i) \qquad B = \sum s(a_i)$$

over all triples in M

over only those a_i whose triples constitute a matching

The alternating bits of B are set as shown for $s(a_1) + s(a_2)$ above.

- Now we are ready to set the binary code words for the sizes $s(b_1)$ and $s(b_2)$:

$$s(b_1) = 2U - B \quad \text{and} \quad s(b_2) = U + B.$$

If you add the sizes for all the elements in $A = \{a_1, a_2, a_3, b_1, b_2\}$, you get a value of $\boxed{4U}$. So the question is whether A can be partitioned into A' and $A - A'$ such that the total size in each partition is $2U$. The answer is YES. We can set $A' = \{a_1, a_2, b_1\}$, $A - A' = \{a_3, b_2\}$

- It is your job to show that if we had started with a NO-instance of 3DM, we would NOT be able to construct a YES-instance of PARTITION with the size assignments as shown above.