

## Homework 8.5 (FAKE) Solutions

*Due: Never**Vinod Vaikuntanathan*

*If every NP-completeness proof had to be as complicated as that for SAT, it is doubtful that the class of known NP-complete problems would have grown as fast as it has.*

*– Garey and Johnson*

This fake homework is intended as a study guide covering the material in lectures 20 and 21 (NP-complete problems).

**Readings:** Sipser, Section 7.5. Also (optionally) see Garey and Johnson’s book, “Computers and Intractability: a Guide to NP-Completeness”.

**Problem 1:** In class, we covered constructions reducing 3SAT directly to four other problems:

- CLIQUE =  $\{\langle G, k \rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$ ,
- HAMPATH =  $\{\langle G, s, t \rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$ ,
- SUBSET-SUM =  $\{\langle S, t \rangle \mid S = \{x_1, \dots, x_k\} \text{ and for some } \{y_1, \dots, y_\ell\} \subseteq \{x_1, \dots, x_k\}, \text{ we have } \sum y_i = t\}$ , and
- 3-DIMENSIONAL-MATCHING =  $\{\langle A, B, C, M \rangle \mid A, B, C \text{ are disjoint sets of size } n, M \subseteq A \times B \times C, \text{ a set of acceptable triples, such that } \exists M' \subseteq M, |M'| = n, \text{ and each element of } A, B, C \text{ appears exactly once in } M'\}$ .

In this problem, we propose variations on the constructions that were presented and ask you whether they work or not, and why.

1. We modify the construction reducing 3SAT to CLIQUE by adding an edge between each pair of nodes in the same triple, unless the pair is contradictory (e.g.,  $x$  and  $\bar{x}$ ).

**No!** This reduction maps a Boolean formula  $\phi$  to a graph  $G$  and a value  $k$ . (See page 252 of Sipser.) In the original reduction, we set  $k$  to be the number of clauses in  $\phi$ . Adding these additional edges to  $G$  might render a  $k$ -clique for an unsatisfiable formula, because it would then be possible to “skip” a clause when selecting nodes for the  $k$ -clique (i.e., take two true literals from one clause, but skip a clause with three false literals). Obviously, this is not acceptable.

2. In the construction reducing 3SAT to HAMPATH, we constructed a diamond for each variable. The horizontal row contains  $3k + 1$  nodes in addition to the two nodes on the ends belonging to the diamond (here  $k$  is the number of clauses in  $\phi$ ). Now, we try to make the reduction more efficient by cutting out the “separator” nodes in the diamond, reducing the size of the horizontal row by  $\frac{1}{3}$ .

**No!** Actually, a similar error appeared in the first printing of Sipser’s book. If your textbook was printed before 2001, you might want to check the HAMPATH proof in a friend’s copy. So, why would removing the separator nodes be a problem? The separator nodes help force the zig-zag (or zag-zig) traversal of each diamond, which corresponds to a true or false setting of that variable. If they were not present, a situation, like that illustrated in Figure 7.19 on page 267, might occur. Exercise for the reader: can you think of an example where this would happen? In such a case, it might be possible for a Hamiltonian graph to exist without  $\phi$  being satisfiable.

3. In the construction reducing 3SAT to SUBSET-SUM, we used multisets, by including, for each clause  $C_j$ , two copies of a vector with a 1 in the position corresponding to  $C_j$ . Now we try to avoid the use of multisets by replacing one of these copies with a vector having a 2 in the position corresponding to  $C_j$ .

**No!** This reduction maps a Boolean formula  $\phi$  to an instance of the SUBSET-SUM problem  $\langle S, t \rangle$ , where  $S$  is a set of numbers and  $t$  is the target sum. (See page 269 of Sipser; this reduction will be covered in lecture on Monday.) Let's first understand how multiple copies of the same number might be placed into  $S$ . The construction in Sipser places two numbers  $y_i, z_i$  into  $S$  for each variable  $x_i$  in  $\phi$ . These two numbers differ from each other in their low-order decimals (unless, both  $x_i$  and  $\bar{x}_i$  always appear together in any clause – which means we could remove them) – since the low-order decimals of  $y_i$  correspond to the clauses that  $x_i$  makes true, while the low-order decimals of  $z_i$  correspond to the clauses that  $\bar{x}_i$  makes true. Each  $(y_i, z_i)$  differs from  $(y_j, z_j)$  since the construction gives different high-order decimals for each pair with index  $i$ .

The conclusion to draw is that multiple copies of the same number in  $S$  must only come from the  $g_j, h_j$  values that are added for each clause  $c_j$ . In fact,  $g_j = h_j$ ; so could we replace  $g_j$  and  $h_j$  by a single value  $g'_j = 2g_j$ ? No. This would imply that each 3SAT clause must have 3 or 1 true literals, but not 2! The values  $g_j$  and  $h_j$  are “dummy” values provided to ensure that each decimal in  $t$  that represents a clause sums to 3. To achieve this, the construction forces that at least one of the literals in the clause must be true via adding a 1, 2 or 3 from the  $y, z$  range and padding that sum with either 1 or 2 from the “dummy”  $g, h$  range. If the only “padding” provided is a 2, then there might not be an subset that sums to  $t$ , even though  $\phi$  is satisfiable.

4. In the construction reducing 3SAT to RELAXED-3-DIMENSIONAL-MATCHING, we include all the same Truth Assignment triples as before. But we eliminate some of the Clause Satisfaction triples: now, for each clause  $C_j$ , we include only one of the three triples  $(*, b'_j, c'_j)$  that were included before.

**No!** This reduction does not appear in Sipser, but was covered in the recitation. Remember that, the instance of RELAXED-3DM we constructed had two types of triples: one set of triples ensures that the truth assignment is consistent – they are called the *Truth Assignment triples*, and the other set of triples expresses the condition that the input formula is satisfiable/unsatisfiable – these triples are called the *Clause Satisfaction triples*.

Now, since we reduce from 3SAT, for each  $(b'_j, c'_j)$  pair, we have three triples of the form  $(a_{ij}, b'_j, c'_j)$  (this was because each clause contains three variables). For instance, if the clause  $C_j$  is of the form  $x_{i_1} \wedge x_{i_2} \wedge x_{i_3}$ , we have three pairs

$$\begin{aligned} &(a_{i_1 j}, b'_j, c'_j) \\ &(a_{i_2 j}, b'_j, c'_j) \\ &(a_{i_3 j}, b'_j, c'_j) \end{aligned}$$

If the input formula is satisfiable, then there exists a variable in each clause  $C_j$  that makes  $C_j$  evaluates to true. But, *a-priori we do not know which variable makes  $C_j$  true*. If we did, we could include just one of the above three triples, and ensure that the resulting instance of R3DM had a matching. Removing 2 of the 3 triples arbitrarily could lead to the absence of a matching even if the input formula is satisfiable.

**Problem 2:** Two graphs are *isomorphic* if, by renaming the nodes of one, we get a graph that is identical to the other. Formulated as a language, the graph isomorphism problem

$$\text{ISO} = \{ \langle G, H \rangle \mid G \text{ and } H \text{ are undirected graphs and } G \text{ and } H \text{ are isomorphic} \}.$$

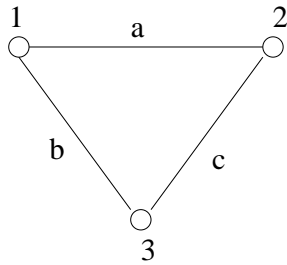
Now, consider a restricted version of graph isomorphism on bipartite graphs – BIPARTITE-ISO.

BIPARTITE-ISO =  $\{(G, H) \mid G \text{ and } H \text{ are undirected bipartite graphs and } G \text{ and } H \text{ are isomorphic}\}$ .

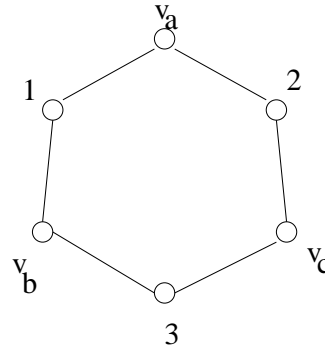
Show that  $\text{ISO} \leq_p \text{BIPARTITE-ISO}$ .

**Solution 2:** Given a pair of graphs  $G$  and  $H$ , we want to construct two bipartite graphs  $G'$  and  $H'$  such that  $G \approx H$  if and only if  $G' \approx H'$ .

Let  $G = (V, E)$  and  $G' = (V', E')$ .  $V'$  is formed by adding  $|E|$  new nodes to  $G$ . That is,  $V' = V \cup \{v_e \mid e \in E\}$ .  $E'$  is formed by replacing each edge  $(i, j)$  of  $G$  by two new edges –  $(i, v_e)$  and  $(v_e, j)$ . That is,  $E' = \{(i, v_e), (v_e, j) \mid (i, j) = e \in E\}$ . (Refer to the figure below for an illustration). The graph  $H'$  is constructed similarly.



$$\begin{aligned} G &= (V, E) \\ V &= \{1, 2, 3\} \\ E &= \{a, b, c\} \end{aligned}$$



$$\begin{aligned} G' &= (V', E') \\ V' &= \{1, 2, 3, a, b, c\} \\ E' &\text{ is as shown} \end{aligned}$$

Any path in  $G$  of length  $\ell$  is manifested in  $G'$  as a path of length  $2\ell$  (since each edge is replaced with two edges). In particular, this means that  $G'$  has no odd cycles, and therefore,  $G'$  is bipartite. So is  $H'$  for the same reason.

Now, suppose there is an isomorphism  $\pi$  from  $G$  to  $H$ . We construct a new isomorphism  $\pi'$  from  $G'$  to  $H'$ .

$$\pi'(v) = \begin{cases} \pi(v) & \text{if } v \in V \\ v_{\{\pi(i), \pi(j)\}} & \text{if } v = v_e \text{ for some edge } e = (i, j) \end{cases}$$

Intuitively,  $\pi'$  maps all the old vertices exactly as  $\pi$  does. The new vertices in  $G'$  are the ones induced by the edges of  $G$ . If  $\pi$  maps an edge  $e$  of  $G$  to an edge  $e'$  of  $H$ , then  $\pi'$  maps the vertex  $v_e$  of  $G'$  to the vertex  $v_{e'}$  of  $H'$ . The proof that this is indeed an isomorphism of  $G'$  to  $H'$  is left as an exercise.

Conversely, suppose there is an isomorphism  $\pi' : G' \rightarrow H'$ . By definition of an isomorphism, we can say that there is an edge between vertices  $u$  and  $v$  in  $G'$  if and only if there is an edge between vertices  $\pi'(u)$  and  $\pi'(v)$  in  $H'$ . Extending this, we can also say that

$$\begin{aligned} &\text{there is a length-2 path between } u \text{ and } v \text{ in } G' \text{ if and only if} \\ &\text{there is a length-2 path between } \pi'(u) \text{ and } \pi'(v) \text{ in } H' \end{aligned} \tag{1}$$

Now, suppose  $\pi'$  is a “good” isomorphism: that is, one which maps vertices  $v \in V \cap V'$  into vertices  $\pi'(v) \in H \cap H'$ . That is, no “old vertex” is mapped to a “newly-created vertex”. Then, the following

is true: But, note that if  $u$  and  $v$  are also vertices of  $G$ , then the existence of a length-2 path between  $u$  and  $v$  in  $G'$  means the existence of an *edge* between  $u$  and  $v$  in  $G$ . Therefore, statement (1) in turn implies that there is an edge between  $u$  and  $v$  in  $G$  if and only if there is an edge between  $\pi'(u)$  and  $\pi'(v)$  in  $H$ . This means,  $G$  and  $H$  are isomorphic via an isomorphism  $\pi$  which is a restriction of  $\pi'$  to vertices of  $G$ .

Now, what is the isomorphism  $\pi'$  is not good? (Convince yourself that there could exist isomorphisms between  $G'$  and  $H'$  such that a vertex  $v$  of  $G'$  is mapped to a vertex of the form  $v_e$  of  $H'$  – this could happen with the figure given above, for instance.) In this case, we can prove that both  $G'$  and  $H'$  are special type of graphs: they are actually unions of disjoint cycles, and there is a one-to-one correspondence between the cycles that form  $G'$  and those that form  $H'$ . What does this tell us about  $G$  and  $H$ ? If  $G'$  is a union of disjoint cycles, so is  $G$  (the component cycles of  $G$  are half the size of the corresponding cycles in  $G'$ ). Now,  $G$  and  $H$  are unions of disjoint cycles too, and are therefore isomorphic.

Thus, we have proved that, if  $G'$  and  $H'$  are isomorphic, then so are  $G$  and  $H$ , and the reduction is complete.

**Problem 3:** The “Set Packing” problem is defined by the language SET-PACKING, which is

$$\{\langle C, k \rangle \mid C \text{ is a collection of finite sets, } k \text{ is a positive integer,} \\ \text{and } C \text{ contains at least } k \text{ disjoint sets}\}.$$

Prove that SET-PACKING is NP-complete, by a reduction from 3-DIMENSIONAL-MATCHING or EXACT-3-COVER.

**Solution 3:** We prove that SET-PACKING is NP-complete by proving that 3-DIMENSIONAL-MATCHING  $\leq_p$  SET-PACKING. (This is a fairly straightforward and “syntactic” reduction).

An instance of 3DM is  $\langle X, Y, Z, M \rangle$ , where  $X, Y, Z$  are sets such that  $|X| = |Y| = |Z| = \ell$  and  $M \subseteq X \times Y \times Z$  is a collection of triples. Create an instance of SET-PACKING  $\langle C, k \rangle$  by setting  $C$  to be the collection of all the triples in  $M$ . i.e,  $C = \{\{x, y, z\} \mid (x, y, z) \in M\}$ .

**Claim 1**  $\langle X, Y, Z, M \rangle \in 3DM$  if and only if  $\langle C, k \rangle \in \text{SET-PACKING}$ .

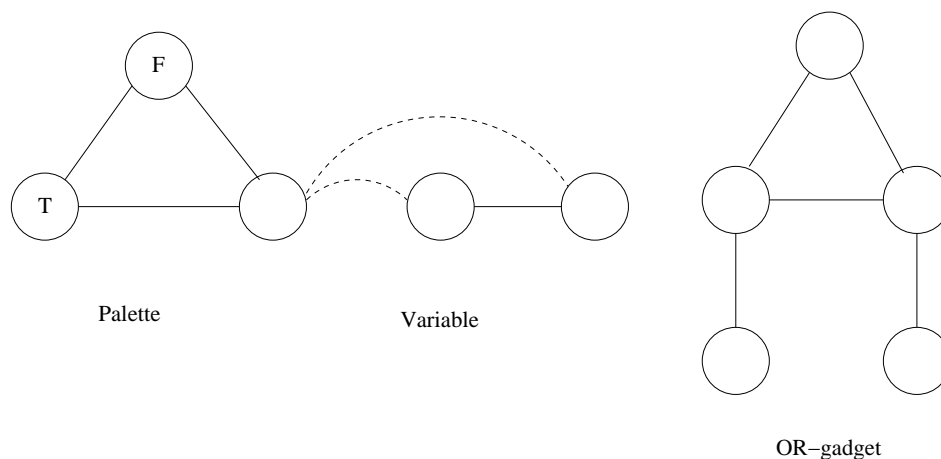
**Proof:** “ $\Rightarrow$ ” Suppose  $\langle X, Y, Z, M \rangle \in 3DM$ . By definition, this means there is a set of triples  $M' \subseteq M$  such that  $M'$  covers each element in  $X \cup Y \cup Z$  exactly once. This means  $C' = \{\{x, y, z\} \mid (x, y, z) \in M'\}$  is a collection of disjoint sets (disjoint, since each element is covered at most once) of cardinality  $\ell = k$  (since there are  $3\ell$  elements to be covered and the cardinality of each set is 3). Therefore,  $\langle C, k \rangle \in \text{SET-PACKING}$ .

“ $\Leftarrow$ ” Suppose  $\langle C, k \rangle \in \text{SET-PACKING}$ . This means, there is a subcollection  $C' \subseteq C$  of disjoint sets such that  $|C'| = k$ . This means,  $M' \stackrel{def}{=} \{\{x, y, z\} \mid \{x, y, z\} \in C'\}$  covers each element in  $X \cup Y \cup Z$  at most once (since the triples in  $M'$  are disjoint). Also, since  $|C'| = k$ ,  $M'$  contains  $k$  triples which, in turn, means that each element is covered exactly once. ■

**Problem 4:** (Sipser 7.27) A **coloring** of a graph is an assignment of colors to its nodes so that no two adjacent nodes are assigned the same color. Let

$$3\text{COLOR} = \{\langle G \rangle \mid \text{the nodes of } G \text{ can be colored with three colors such that} \\ \text{no two nodes joined by an edge have the same color}\}.$$

Show that 3COLOR is NP-complete. (Hint: Use the following three subgraphs.)



**Solution 4:** (editted from Jon Herzog, Spring 2002)

**3-COLOR is in NP.** First, we show that *3-COLOR* is in  $\mathcal{NP}$ , because we can give an actual coloring as a witness. Verifying a coloring means checking the color of two nodes per edge, which takes polynomial time. (Also, representing a 3-coloring takes polynomial number of bits). And since there exists a coloring if and only if the graph is 3-colorable (by definition), we know that the coloring can act as a witness. Hence *3-COLOR* is in  $\mathcal{NP}$ .

(As another way to show this, we can also make a non-deterministic machine that guesses the coloring, then verifies it. This machine *can* accept all 3-colorable graphs, but *must* reject all graphs that cannot be 3-colored. Also, it only takes a polynomial number of steps to decide. So, *3-COLOR* is the language of a non-deterministic decider, and so is in  $\mathcal{NP}$ .)

**3-COLOR is NP-hard.**

To prove that 3-COLOR is NP-hard, we use a reduction from 3SAT. Given a formula  $\phi$  of  $m$  clauses on  $n$  variables  $x_1, x_2, \dots, x_n$ , we construct a graph  $G = (V, E)$  as follows. The set of nodes  $V$  consists of a vertex for each variable, a vertex for the negation of each variable, 5 vertices for each clause, and 3 special vertices. We call these special vertices *TRUE*, *FALSE*, and *RED*. We can not actually produce a partially colored graph  $G$  as the output of this reduction, but we can force each of these vertices to be a different color, by connecting them in a triangle, and thus we can think about their respective colors as:  $T$ ,  $F$ , and  $R$ . The remaining edges of the graph are of two types: “literal” edges that are independent of the clauses, and “clause” edges that depend on the clauses. The literal edges form a triangle between  $x_i, \bar{x}_i$ , and  $RED$  for  $i = 1, 2, \dots, n$ . Thus, we force one of  $x_i, \bar{x}_i$  to be colored *TRUE* and the other *FALSE*.

For example, if the formula  $\phi$  has three literals,  $x_1, x_2$  and  $x_3$ , then the literal nodes and edges will look like this:

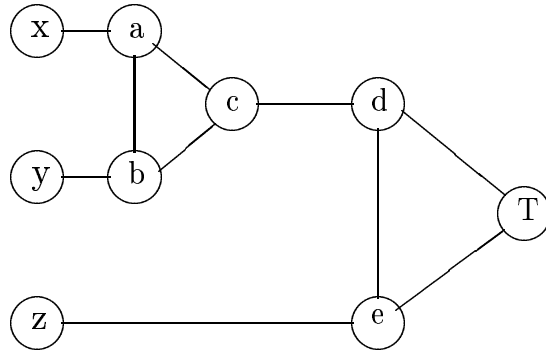
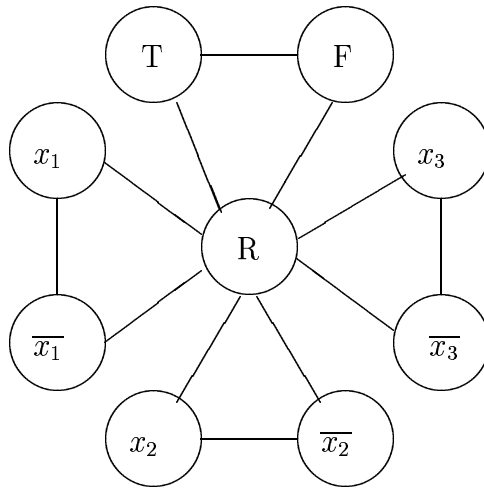


Figure 1: A clause constraint widget for  $(x \vee y \vee z)$ .



Hence we can form a bijection between colorings and assignments:  $x_j$  is true in the assignment for  $\phi$  iff the  $x_j$  node is colored *TRUE*. In this way, we can make a coloring for each assignment, and an assignment for each valid coloring.

Now that our variable gadgets are in place, we must add our clause constraints to the graph. The widget shown in Figure 1 is used to force the condition corresponding to a clause  $(x \vee y \vee z) = \text{true}$ . Each clause requires a unique copy of the 5 vertices that are labeled  $\{a, b, c, d, e\}$  in the figure; they connect as shown to the literals of the clause and the special vertex *TRUE*.

We need to argue that if each of  $x$ ,  $y$  and  $z$  is colored *TRUE* or *FALSE* (as forced by our variable gadgets), then this subgraph is 3-colorable iff one of  $x$ ,  $y$ , and  $z$  is colored *TRUE*.

Consider the graph in Figure 1. Now, we can assume that the nodes  $x$ ,  $y$  and  $z$  are all colored either *T* or *F*, and that the *T* node is colored *T*. We can think of the right-most node being colored *T* as the corresponding clause being true; the right-most node in each clause gadget will actually be the special vertex that corresponds to *TRUE* from our original special triangle. Using this same node will also supply the AND condition needed to unite the clause widgets. We want to show that the widget is 3-colorable iff at least one of  $x$ ,  $y$  and  $z$  is colored *T*. We can show one direction by

simply giving colorings:

$x$	$y$	$z$	$a$	$b$	$c$	$d$	$e$
$T$	$T$	$T$	$F$	$R$	$T$	$F$	$R$
$T$	$T$	$F$	$F$	$R$	$T$	$F$	$F$
$T$	$F$	$T$	$F$	$R$	$T$	$F$	$F$
$T$	$F$	$F$	$F$	$R$	$T$	$F$	$F$
$F$	$T$	$T$	$T$	$R$	$F$	$R$	$R$
$F$	$T$	$F$	$T$	$F$	$R$	$F$	$R$
$F$	$F$	$T$	$T$	$R$	$F$	$R$	$F$

As for the other direction, suppose that  $x$ ,  $y$ , and  $z$  are all colored  $F$ . Then note that  $a$  and  $b$  have to be colored differently, and neither can be colored  $F$ . Hence,  $c$  must be colored  $F$ . This means that  $d$  and  $e$  must be colored differently, but neither can be colored  $F$ . And since neither can be colored  $T$  because of the right-most node, we have a contradiction.

Now that we've learned the variable and clause gadget tricks, let's summarize how to map a boolean formula  $\phi$  into a graph  $G$ :

1. First, we write down the special  $T$ ,  $F$ , and  $R$  nodes and connect them into a triangle.
2. Then for each literal  $x_i$ , we write down a node for  $x_i$  and  $\overline{x}_i$ , and make a triangle between those two nodes and  $R$ .
3. For each clause, we make a new widget. The  $x$ ,  $y$ , and  $z$  nodes of the widget will actually be the appropriate literal nodes. For example, for a clause of the form  $(x_1 \wedge \overline{x}_2 \wedge x_3)$ , we would write down a new widget where the  $x$  node is really the  $x_1$  node,  $y$  node is really the  $\overline{x}_2$  node, and  $z$  node is really the  $x_3$  node. In every new widget, the  $T$  node is, in fact, the  $T$  node we wrote down in the first step.

Now, to show that there is a 3-coloring iff  $\phi$  is satisfiable:

- Suppose that there is a 3-coloring. As we argued in the first part, this means that each  $x_i$  node and each  $\overline{x}_i$  node is colored either  $T$  or  $F$ . And since each widget is 3-colorable, this means that for each widget, at least one of the  $x$ ,  $y$  and  $z$  nodes is colored  $T$ .

So, map the coloring to an assignment as such: if  $x_i$  is colored  $T$ , then make  $x_i$  true in the assignment. Otherwise,  $\overline{x}_i$  is colored  $T$ , and we make  $x_i$  false in the assignment. We know that this assignment is satisfying, because each widget is 3-colorable. This means that for each clause, some  $x$ ,  $y$  or  $z$  node is colored  $T$ , which means that the assignment makes some literal (or converse) in each clause true. Hence, the clause is satisfied.

- Now, suppose that there is a satisfying assignment. Then we can color the literal nodes according to the assignment. Because the assignment is satisfying, the assignment makes at least one literal (or converse) of each clause true. Hence, for each widget, at least of the  $x$ ,  $y$  or  $z$  nodes will be colored  $T$ . Hence each widget is 3-colorable in a way consistent with the literal nodes. Hence, the entire graph is 3-colorable.

Thus,  $3\text{-SAT} \leq_P 3\text{-COLOR}$ . And since  $3\text{-COLOR}$  is in NP, we know that  $3\text{-COLOR}$  is NP-complete.