Informatics 2 – Introduction to Algorithms and Data Structures

Solutions for tutorial 9

1. (a) We will show that SAT is in NP, 3-SAT can be verified exactly the same way. Our "solution/certificate" for a SAT instance $\phi = C_1 \wedge \ldots C_m$ will be an assignment to the logical variables $\{x_1, \ldots, x_n\}$ - note this can be represented as a binary string of length n (so polynomial in the size of the input formula ϕ). Let the assignment be $b \in \{0,1\}^n$, with b_i being the assignment for variable x_i.

   To do the verification against ϕ, our algorithm needs to consider each clause C_j in turn, and then for each of the literals in C_j, check whether it is satisfied against b - as long as one literal is satisfied by b, then C_j is satisfied. It takes $O(|C_j|)$ time to check C_j. To check *all* C_j are satisfied takes $O\left(\sum_{j=1}^{m} |C_j|\right)$ in total, which is polynomial-time, in fact *linear*, in the size of our input instance ϕ.

   Hence SAT, 3-SAT are both in NP.

   (b) We have an instance $\phi = C_1 \wedge \ldots \wedge C_m$ of SAT, over boolean variables $\{x_1, \ldots, x_n\}$. We will assume that no C_j includes any *complementary pair* of literals $x_i, \bar{x_i}$, as any such clause is trivially satisfied (and can be removed from the list of clauses without changing the Satisfiability). These can be easily detected (and clauses removed) in polynomial-time.

   For any clause C_j such that $|C_j| = 1$ (say C_j is $\ell_{j,1}$), we will create *two* dummy variables $y_{j,1}, y_{j,2}$, and then we will replace C_j by the following four clauses:

$$C_{j,1} = (\ell_{j,1} \vee y_{j,1} \vee y_{j,2}) \qquad C_{j,2} = (\ell_{j,1} \vee \bar{y_{j,1}} \vee y_{j,2})$$
$$C_{j,3} = (\ell_{j,1} \vee y_{j,1} \vee \bar{y_{j,2}}) \qquad C_{j,4} = (\ell_{j,1} \vee \bar{y_{j,1}} \vee \bar{y_{j,2}})$$

   Observe that regardless of which 0/1 values are given to $y_{j,1}, y_{j,2}$ the "dummy literals" will be both 0 in one of the four clauses, enforcing $\ell_{j,1}$ to be satisfied, which is what we require.

   For any clause C_j such that $|C_j| = 2$ (say C_j is $(\ell_{j,1} \vee \ell_{j,2})$), we will create *one* fresh dummy variable y_j, and replace C_j by the following two clauses:

$$C_{j,1} = (\ell_{j,1} \vee \ell_{j,2} \vee y_j), C_{j,2} = (\ell_{j,1} \vee \ell_{j,2} \vee \bar{y_j}).$$

   For any clause C_j such that $|C_j| = 3$, we leave that clause as it is.

   Finally, consider any clause C_j with $|C_j| > 3$. For these clauses, we need to add $|C_j| - 3$ new dummy variables $y_{j,1}, \ldots y_{j,|C_j|-3}$. We then replace C_j with the following clauses $C_{j,1}, \ldots, C_{j,|C_j|-2}$ defined as follows:

$$C_{j,i} = \begin{cases} (\ell_{j,1} \vee \ell_{j,2} \vee y_{j,1}) & i = 1 \\ (\bar{y}_{j,i} \vee \ell_{j,i+1} \vee y_{j,i+1}) & i = 2, \ldots, |C_j| - 3 \\ (\bar{y}_{j,|C_j|-2} \vee \ell_{j,|C_j|-1} \vee \ell_{j,|C_j|}) & i = |C_j| - 2 \end{cases}$$

Then assignment b will satisfy $C_j \Leftrightarrow$ we can extend this assignment to assign values to all the dummy variables to satisfy all of these $|C_j| - 2$ clauses.

Notice that for this $|C_j| > 3$ case, we will add $\leq |C_j|$ variables, and we also expand the size of the clausal representation by at most a factor of 3 (counting total number of literals, not individual clauses).

We have created an instance of 3-SAT of total size at most 3 times our original problem. Each of the conversions to 3-CNF are methodological, and can be done in time linear in the size of C_j. Hence SAT $\leq_P$ 3-SAT.

2. In this question we consider whether the decision version (COIN) of the "coin changing" problem is in P, and/or in NP.

   (a) The proposed algorithm to solve the decision version of (COIN) suggests running the dynamic programming algorithm to evaluate the *minimal* number of coins for value v in the coin system $\{c_1, \ldots, c_k\}$. However the use of the DP algorithm as a starting point means it can't be polynomial time.

   Numbers entered as input parameters to a computational problem are written in binary format (or some related format such as decimal, or hexadecimal ... but never unary, that's not sensible). So for example, the value $v = 2^5 = 32$ is written with 6 bits as 100000, the value $v = 2^{10} = 1024$ is written with 11 bits as 10000000000, and the value $v = 2^{20} = 1048576$ is written with 21 bits as 10000000000.

   The length of these representations (whether the original decimal, or the slightly longer binary) is tiny in comparison to the *value* of the number.

   When we run the dynamic programming for coin-changing, we create a table of dimensions $(k+1) \cdot (v+1)$, and the loop to fill the table is $\Theta(k \cdot v)$.

	0	1	...	...	...	v
$\emptyset$	0	0	0	0	0	0
$\{c_1\}$	$\cdot$	$\cdot$	...	...	...	$\cdot$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\{c_1, \ldots, c_k\}$	$\cdot$	$\cdot$	...	...	...	$\cdot$

   But this table size (and running-time) is exponential in $\lg(v)$. If we'd had $v = 2^{20}$, the table would have had dimensions $1048576 \cdot (k+1)$, for example. Hence this approach can't give us a polynomial-time algorithm.

   Algorithms which are polynomial-time in "everything except the numbers" (like this one) are sometime called *pseudopolynomial.*

   (b) We are next asked to consider whether it is possible to *verify* a "certificate" of COIN$(c_1, \ldots, c_k; v; h)$ being True in polynomial-time (without any directive of the algorithm for checking).

   This question is a bit more nuanced. We have been talking about a "collection of coins" in referring to a solution for COINS, with the default being a multiset.

   Option 1: If we represent the "collection of coins" as a multiset, then the implication is that we would list the same coin multiple times. This causes a problem for input data $c_1, \ldots, c_k; v$ where the target value v is much greater compared to any of the coin sizes. For example, considering $\{1, 5\}$ as our coin set, but some very large $v = 5^n$ (say), we would need 5^{n-1} coins in any solution to make v,

This number of coins is *exponential* in the size of our binary representation for v (which only requires $\lceil n \cdot \lg(5) \rceil$ bits to input). Hence even before considering the *checking* of the multiset, its representation prohibits it from being polynomial-time verifiable.

Option 2: However, we have been ambiguous about how we represented our multiset of coins for the solutions to coin-changing. A more appropriate way to represent the solutions would be a list of pairs $(c_i, n_i), i = 1, \ldots, k$, with n_i being the number of c_i coins to be taken in the solution.

Note that $n_i \leq v$ for every i (at a minimum) so with this representation, then the size of a "certificate" becomes polynomial in the size of the input parameters.

Then we need to do the verification. This is done by an arithmetic calculation where we multiply $c_i \times n_i$ (multiplication is quadratic in the lg-size of the two numbers) for each i, and then add them together (linear in the lg-size of the number being added).

So the checking is also polynomial-time with this more sensible representation for solutions.

3. We are considering the derandomization algorithm for $\frac{7}{8}m$ *expected number of satisfied clauses* for 3-CNF that was presented in Lecture 26. The "Live Discussion" for week 6 has an example of this process being executed on a specific Φ.

   (a) We construct a specific assignment to satisfy $Y \geq \frac{7}{8}9$, ie at least 8, of the clauses in the following Φ, where Y is the r.v. measuring the number of satisfied clauses. We have 4 logical variables, so our assignments are from $\{0,1\}^4$.

$$\begin{aligned}\Phi &= (x_1 \vee x_2 \vee x_3) \wedge (\bar{x_1} \vee \bar{x_2} \vee \bar{x_3}) \wedge (\bar{x_1} \vee x_2 \vee x_3) \wedge \\ &\quad (x_1 \vee \bar{x_2} \vee \bar{x_3}) \wedge (x_1 \vee \bar{x_2} \vee x_4) \wedge (x_2 \vee x_3 \vee \bar{x_4}) \wedge \\ &\quad (\bar{x_1} \vee \bar{x_3} \vee \bar{x_4}) \wedge (\bar{x_2} \vee \bar{x_3} \vee x_4) \wedge (\bar{x_1} \vee x_3 \vee x_4).\end{aligned}$$

variable x_1: We first consider variable x_1, and the two options $x_1 \leftarrow 0$ and $x_1 \leftarrow 1$. To compute $Exp_0 = \mathsf{E}[Y \mid x_1 \leftarrow 0]$, the expected number of satisfied clauses *conditional on x_1 being 0*, we notice that Φ has

- 4 clauses containing the negative literal $\bar{x_1}$
- 3 clauses containing the positive literal x_1
- 2 clauses not involving this variable at all.

By setting $x_1 \leftarrow 0$, we satisfy the $\bar{x_1}$ clauses immediately (with probability 1), we delete the x_1 literal from the clauses that had contained it (henceforth they only have two "active" literals and their probability of being satisfied drops from $\frac{7}{8}$ to $\frac{3}{4}$) ... and the probability of the two uninvolved clauses being satisfied remains at $\frac{7}{8}$. This gives

$$\mathsf{E}[Y \mid x_1 \leftarrow 0] \;=\; 4 + 3\frac{3}{4} + 2\frac{7}{8} \;=\; 8.$$

To compute $Exp_1 = \mathsf{E}[Y \mid x_1 \leftarrow 1]$, we just need to note that the circumstances for the positive literals (3 of these) and negative literals (4 of these) are reversed, hence the value Exp_1 can be computed as

$$\mathsf{E}[Y \mid x_1 \leftarrow 1] \;=\; 3 + 4\frac{3}{4} + 2\frac{7}{8} \;=\; 7.75.$$

The rules of the derandomization algorithm require us to choose the assignment to x_1 which maximizes the expectation, hence we assign $x_1 \leftarrow 0$. Propagating this to Φ, we get:

$$\begin{aligned}
\Phi' \quad = \quad & (\cancel{x_1 \vee}\ x_2 \vee x_3) \wedge\ (\bar{x_1} \vee \bar{x_2} \vee \bar{x_3}) \wedge\ (\bar{x_1} \vee x_2 \vee x_3) \wedge \\
& (\cancel{x_1 \vee}\ \bar{x_2} \vee \bar{x_3}) \wedge\ (\cancel{x_1 \vee}\ \bar{x_2} \vee x_4) \wedge (x_2 \vee x_3 \vee \bar{x_4}) \wedge \\
& (\bar{x_1} \vee \bar{x_3} \vee \bar{x_4}) \wedge\ (\bar{x_2} \vee \bar{x_3} \vee x_4) \wedge\ (\bar{x_1} \vee x_3 \vee x_4)
\end{aligned}$$

We already have 4 clauses that are definitely satisfied.

Variable x_2**:** Next we consider literal x_2, and the two options $x_2 \leftarrow 0$ and $x_2 \leftarrow 1$. We only have five "active" clauses to consider at this point, three of size 2 and two of size 3.

To consider the effect of setting $x_2 \leftarrow 0$, we notice that x_2 is negative in 3 of the remaining clauses, and positive in one length-2 clause and positive in one length-3 clause. By setting $x_2 \leftarrow 0$ we immediately satisfy the 3 clauses containing $\bar{x_2}$ (with probability 1) regardless of their length, but we need to delete the positive literal x_2 from the two active clauses which contain it ... then one clause ends up with a single remaining literal, with the probability of being satisfied dropping to $\frac{1}{3}$ and the length-3 clause which contains x_2 will reduce to length 2, and the probability of being satisfied drops from $\frac{7}{8}$ to $\frac{3}{4}$. Hence

$$\mathsf{E}[Y \mid x_1 \leftarrow 0, x_2 \leftarrow 0] \ = \ \mathbf{4} + 3 + 1\frac{1}{2} + 1\frac{3}{4} \ = \ 8.25,$$

where we use bold to indicate that the initial term $\mathbf{4}$ is from the previously satisfied clauses.

For $\mathsf{E}[Y \mid x_1 \leftarrow 0, x_2 \leftarrow 1]$, I will just observe that in the <u>uniform random model</u> for assignments, the assignments to x_2, x_3, x_4 (conditional on our prior decision $x_1 \leftarrow 0$) are equally split between $x_2 \leftarrow 0$ and$x_2 \leftarrow 1$. Hence,

$$\mathsf{E}[Y \mid x_1 \leftarrow 0] \ = \ \frac{\mathsf{E}[Y \mid x_1 \leftarrow 0, x_2 \leftarrow 0] + \mathsf{E}[Y \mid x_1 \leftarrow 0, x_2 \leftarrow 1]}{2}.$$

The value of $\mathsf{E}[Y \mid x_1 \leftarrow 0]$ was 8 and the value of $\mathsf{E}[Y \mid x_1 \leftarrow 0, x_2 \leftarrow 0]$ is 8.25 ... given the equation above, we don't need to compute $\mathsf{E}[Y \mid x_1 \leftarrow 0, x_2 \leftarrow 1]$ to know it is less than $\mathsf{E}[Y \mid x_1 \leftarrow 0, x_2 \leftarrow 0]$.

Therefore we choose $x_2 \leftarrow 0$. Propagating this to Φ, we get:

$$\begin{aligned}
\Phi' \quad = \quad & (\cancel{x_1 \vee x_2 \vee}\ x_3) \wedge\ (\bar{x_1} \vee \bar{x_2} \vee \bar{x_3}) \wedge\ (\bar{x_1} \vee x_2 \vee x_3) \wedge \\
& (\cancel{x_1 \vee}\ \bar{x_2} \vee \bar{x_3}) \wedge\ (\cancel{x_1 \vee}\ \bar{x_2} \vee x_4) \wedge (\cancel{x_2 \vee}\ x_3 \vee \bar{x_4}) \wedge \\
& (\bar{x_1} \vee \bar{x_3} \vee \bar{x_4}) \wedge\ (\bar{x_2} \vee \bar{x_3} \vee x_4) \wedge\ (\bar{x_1} \vee x_3 \vee x_4)
\end{aligned}$$

We now have 7 clauses definitely satisfied, and just 2 remaining clauses to consider.

Variable x_3**:** Next we consider literal x_3, and the two options $x_3 \leftarrow 0$ and $x_3 \leftarrow 1$. We only have two "active" clauses to consider at this point, one of size 1 and one of size 2. Both of these clauses have x_3 as a positive literal.

To consider the effect of setting $x_3 \leftarrow 0$, we note that this will cause clause 1 to fail, and will reduce the number of active literals of clause to drop to 1. Hence the expectation conditional on fixing x_3 as 0 will have little except the already guaranteed 7:

$$\mathsf{E}[Y \mid x_1 \leftarrow 0, x_2 \leftarrow 0, x_3 \leftarrow 0] \ = \ \mathbf{7} + 0 + 1\frac{1}{2} \ = 7.5.$$

The effect of fixing $x_3 \leftarrow 1$ is obviously to make both "active" clauses be satisfied, giving a overall value of 9.

Hence we choose $x_3 \leftarrow 1$.

At this point we have *already* satisfied ał9 clauses.

Hence we may choose either value for x_4, doesn't matter which.

Overall assignment is $x_1 \leftarrow 0, x_2 \leftarrow 0, x_3 \leftarrow 1, x_4 \in \{0, 1\}$.

(b) If we are are to attempt the same process on CNF formulae which are not CNF, there first thing to do is to observe is that the initial expectation calculation $\mathsf{E}[Y] = \frac{7}{8}m$ no longer fits. We can still use *linearity of expectation*, however - we need to compute the number of clauses m_k of length k in Φ, for every $k = 1, \ldots, n$. Then the expected number of satisfied clauses $\mathsf{E}[Y]$ is

$$\mathsf{E}[Y] \;=\; \sum_{k=1}^{n} m_k(1 - \tfrac{1}{2^k})$$

After that, we can carry out a derandomization which will be guaranteed to build a specific assignment which satisfies *at least as* many clauses as $\mathsf{E}[Y]$. As in the 3-CNF cases, we should examine the x_i in some order, choosing a fixed value for some x_i at each step based on which gives that larger conditional expectation.

While calculating the $\mathsf{E}[Y \mid x_1 = b_1, \ldots, x_j = b_j]$ values during this process, we will be referring to a Φ' formula which will have clauses of varying sizes. Therefore, our calculations will need to add values $(1 - \frac{1}{2^k})$ for $k > 3$ - however, it doesn't make the calculations any less feasible than for 3-CNF, and the derandomization can still be carried out in polynomial-time.

The only difference is that we won't necessarily get an assignment satisfying $\geq \frac{7}{8}m$ clauses, because the initial expectation might not have been as high as that (especially if Φ has a lot of 1-literal and/or 2-literal clauses).

4. Relationship between Vertex Cover and Independent Set problems.

- $\mathcal{I}$ is an *Independent Set* of G if for every $u \in \mathcal{I}, v \in \mathcal{I} \setminus \{u\}$, that $(u, v) \notin E$.
- $\mathcal{K}$ is a *Vertex Cover* of G if for every $e = (u, v) \in E$, either $u \in \mathcal{K}$ or $v \in \mathcal{K}$.

(a) **proof:** By definition, the set $\mathcal{I}$ is an Independent set if (and only if) for every $u \in \mathcal{I}, v \in \mathcal{I} \setminus \{u\}$, that $(u, v) \notin E$.

This is the case if and only if for every $(u, v) \in E$, at least one of u, v is *not* in $\mathcal{I}$.

This is the case if and only if if for every $(u, v) \in E$, either $u \in V \setminus \mathcal{I}$ or $v \in V \setminus \mathcal{I}$.

This is the case (by definition) if and only if $V \setminus \mathcal{I}$ is a Vertex Cover for G.

Implications for the two decision problems: The Independent Set Decision question asks *whether G has an Independent Set $\mathcal{I}$ of size* $|\mathcal{I}| \geq k$. As shown above this is the case if and only if *G has a Vertex Cover $V \setminus \mathcal{I}$ such that* $|\mathcal{I}| \geq k$. However, $|\mathcal{I}| \geq k \Leftrightarrow |V \setminus \mathcal{I}| \leq (n - k)$.

Therefore the equivalent statement is that *G has a Vertex Cover $\mathcal{K}$ such that* $|\mathcal{K}| \leq (n - k)$.

The equivalence between the two problems gives a very straightforward "reduction" from Independent Set to Vertex Cover, *and vice versa.* This is a really simple reduction, where the graph stays exactly the same, and the only change is to the size parameter (converting via $n - k$ in both reductions).

Therefore, Independent Set is NP-complete ⇔ Vertex Cover is NP-complete.

It's quite rare to have "reductions" which work in both directions in fact.

Note that we gave a proof of Independent Set being NP-complete in Lecture 25, conditional on 3-SAT being NP-complete. The relationship demonstrated in this question allows is to further infer that Vertex Cover is NP-complete.

(b) The relationship proved in (a) tells us that the Decision versions of the Independent Set and Vertex Cover are equivalent from the point of view of polynomial-time computation. If we have an algorithm to solve one of these problems, it can solve the other.

However, suppose we have an *approximation algorithm* for one of the problems, say Vertex Cover, and suppose this algorithm has an approximation ration of α, for $\alpha > 1$. This means that the algorithm is guaranteed to return a value ℓ satisfying $\ell \leq \alpha \cdot OPT_{VC}(G)$, where $OPT_{VC}(G)$ is the minimum size of a VC for G.

Let's now consider whether we can infer anything about the value $n - \ell$, when interpreted in relation to the optimum Independent Set for G (which we know is has size $n - OPT_{VC}(G)$).

The approximation guarantee for Vertex Cover guarantees that when we consider the implications for Independent Set, we find

$$n - \ell \;\geq\; n - \alpha \cdot OPT_{VC}(G) \;=\; (n - OPT_{VC}(G)) - (\alpha - 1)OPT_{VC}(G)$$

What we would really like $n-\ell \geq \frac{1}{\beta}(n - OPT_{VC}(G))$ for some $\beta > 1$ (preferably with some nice relationship to the α), however the extra $-(\alpha-1)OPT_{VC}(G)$ will make this impossible.

As a concrete example, suppose our α of the Vertex Cover approximation is $\alpha = 2$. Then when we take $n - \ell$ for Independent Set, we have the bound

$$n - \ell \;\geq\; (n - OPT_{VC}(G)) - OPT_{VC}(G)$$

However, there may be graphs where $OPT_{VC}(G)$ is $n/2$ or even greater. In that case, the right-hand side above will be drastically affected by the extra $-OPT_{VC}(G)$, and $n - \ell$ may be arbitrarily close to 0.

Hence a nice constant approximation for Vertex Cover will not necessarily converted into a constant (or any!) approximation for Independent Set, or vice versa.

The issue here is the $-$ in the conversion between the two problems: subtraction does not preserve approximation.

5. 3-Col problem for a given input graph $G = (V, E)$.

(a) The first question asks us to show that 3-Col belongs to NP. Assume the 3 colours are G, R, B.

Our "certificate" for this problem is simple: a list of vertex-colour pairs, one for each vertex $v \in V$. This is $\Theta(n)$ wrt to the size of the vertex set.

This can be checked in $\Theta(n + m)$ as follows, assuming we have access to an Adjacency list structure Adj, with $Adj[v]$ being the list of adjacent vertices to v.

To start our checking, we initialise two arrays of length n, array C (entries initialised to "-") and array N (entries initialised to 0).

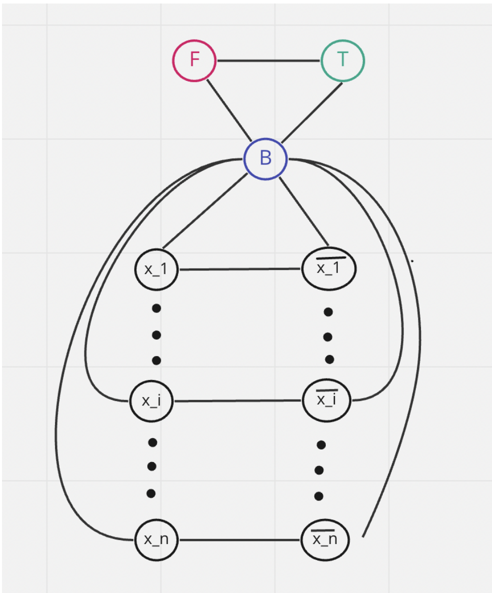
We scan the list of vertex-colour pairs, and when reading each (v, c), we update $C[v] \leftarrow c$, $N[v] \leftarrow N[v] + 1$, until the entire sequence of pairs is read.

Next we check through the array N in $O(n)$ time to make sure every cell has value 1. If not, we return False.

Then we iterate through the Adjacency list: for each node j, we iterate through its list of neighbours w, checking that $C[v] \neq C[w]$. If any of these checks fail, we return False. The entire Adjacency list can be checked in $\Theta(n + m)$ time.

if we finish all the checks without finding a violation, return True.

(b) We refer to the diagram in discussing this.

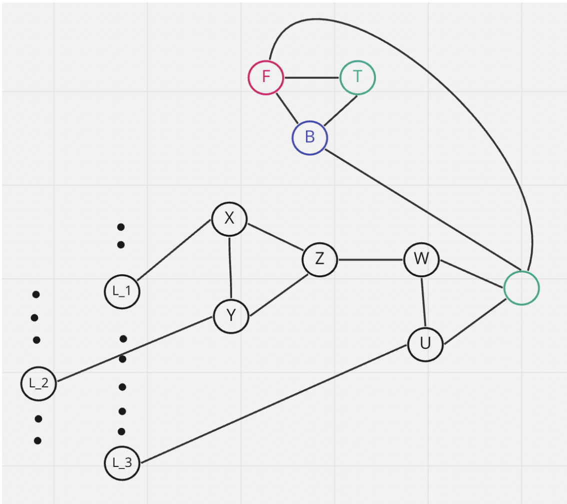


Consider the $x_i, \bar{x}_i$ nodes for any $i = 1, \ldots, n$. These nodes are set up to form a triangle with the B node of the "central triangle". This means that neither x_i nor $\bar{x}_i$ can take the same colour as B. Also, the edge connecting $(x_i, \bar{x}_i$ prohibits the opposing literals form taking the same colour: so we are guaranteed that x_i is either "T's colour" (green in pictures) or "F's colour" (red in pictures), and that $\bar{x}_i$ has the other colour. Note that either of these is possible, there is no bias.

This encodes the two possible "truth assignments" for x_i.

There is the same triangle set-up (with the same B) for every x_i; hence every proper 3-colouring of the overall graph will encode some truth assignment for the logical variables, and all truth assignments are feasible.

(c) To prove this correspondence, it is helpful to label the nodes of the 6-vertex gadget (except the rightmost one, which is fixed to green/"T's colour").



We will show that we can extend a proper 3-colouring (on the truth-setting subgraph) to a proper 3-colouring including these 6 vertices $\Leftrightarrow$ at least one of the literals L_1, L_2, L_3 is green ("T's colour").

IF: First of all suppose that at least one of L_1, L_2, L_3 is green.
If L_1 is green, then we can set $c(X) = F$ (red), $c(Z) = T$ (green), $c(Y) = B$ (blue) to satisfy the left triangle, and it also is proper for the edges (L_1, X) and (L_2, Y), *regardless* of the truth value for L_2 (as the adjacent node Y is getting blue). We can then set $c(W) = F$, $c(U) = B$ and these will work for the right triangle regardless of L_3's value, as U is getting blue.
If L_2 is green then we can make an exactly symmetric argument, just swapping the assignment to X with Y.
If L_3 is the only green literal, then in the left triangle, we are forced to use colours T/green and B/blue on vertices X and Y. Then Z is forced to be F/red ... and subsequently W is forced to be B/blue. Then in the right triangle we have a green and a blue and U is forced to be red . This is consistent with L_3 being T/green, so we have a 3-colouring.

ONLY IF: Suppose that none of at least one of L_1, L_2, L_3 is T/green
We will derive a contradiction to the existence of a 3-colouring. The initial reasoning follows the argument when we know both L_1 and L_2 are F/red,, with the conclusions that vertices X and Y use T/green and B/blue, that Z must be F/red, W must be B/blue, and U is forced to be red.
However, L_3 is **not** true, so also has the colour F/red, and hence we have an impossible constraint along edge (L_3, U).
We have shown that if all 3 literals are False, then there cannot be a 3-colouring of the "truth-setting graph" with the C_j gadget attached.

(d) We hook-up a "6-vertex gadget" (with fresh vertices) for each clause C_j of the formula Φ. Note that we end up with $3 + 2n$ nodes for the "truth setting" subgraph of (b), plus 6 fresh nodes for every clause C_j, so we have $3 + 2n + 6m$ nodes in total, which guarantees this is a polynomial-sized reduction.

Mary Cryan, 10th March 2025