Tutorial 9: 3Sat and **NP**-completeness

1 3Sat and IndepSet

We know that IndepSet is **NP**-complete since in lectures we saw that

1. IndepSet $\in$ **NP**, and
2. 3Sat $\leq_{\mathbf{P}}$ Clique $\leq_{\mathbf{P}}$ IndepSet.

Show (directly) that 3Sat $\leq_{\mathbf{P}}$ IndepSet.

1.1 Solution

Let φ be a 3SAT formula over n variables with m clauses. We will construct a graph for input to the IndepSet problem as follows.

Create a gadget for each of the m clauses containing one vertex for each literal in the clause. Each pair of vertices from the same clause will have an edge between them. Furthermore, we add edges between contradictory literals from different clauses. That is, if clause A contains x_i and clause B contains $\neg x_i$, we add an edge between the vertex for x_i in clause A and the vertex for $\neg x_i$ in clause B.

Suppose that φ has a satisfying assignment. Then for each clause, we can select a literal in the clause which is true in this assignment, yielding a set S of m literals. These literals all came from different clauses, so they correspond to vertices of different gadgets in G. Therefore, the only edges which may be between vertices of S are those between contradicting literals in different gadgets. However, since S comes from a valid satisfying assignment, there is no variable x_i such that $x_i \in S$ and $\neg x_i \in S$, so S is an independent set in G of size m.

Conversely, suppose that G has an independent set S of size at least m. Since each gadget is a clique, S contains at most one vertex from each gadget. Since $|S| \geq m$ and there are m gadgets, S contains exactly one vertex from each gadget. Let A be the set of literals corresponding to the vertices in S. A is a "partial" assignment because it may not contain every variable. Extend A to a full assignment by arbitrarily assigning the variables which do not otherwise appear in S. Note that this is a valid assignment, because S is an independent set and any contradictory literals have edges between them in G. Also, A satisfies φ since it contains at least one literal from each clause.

2 3Sat and DomSet

A *dominating set* in a graph $G = (V, E)$ is a set $S \subseteq V$ of vertices such that every vertex in V is adjacent to at least one vertex in S in the graph G.[1]

[1] Is this the same as a *vertex cover* of G? You should be able to construct a small example that convinces you the two notions are quite different.

In the DomSet problem, we are given as input a graph $G = (V, E)$ and a positive integer k and we must determine whether G has a dominating set of size at most k. For this question, you will show that DomSet is **NP**-complete.

(i) Prove that DomSet $\in$ **NP**.

(ii) Prove that 3Sat $\leq_{\mathbf{P}}$ DomSet.

2.1 Solution

(i) We require as certificate a dominating set S of the graph G of size at most k. A polynomial-time verifier can first verify that $|S| = k$ and then simply go through all the vertices in V and check all its neighbours to verify that at least one of them is in S. When (G,k) is a `Yes` input, then there is at least one dominating set of size k in G that causes the verifier to return `Yes`. When (G,k) is a `No` input, then for every set S of size k there must be some vertex in V not adjacent to any vertex in S and the verifier always returns `No`.

(ii) Given an instance φ of the 3Sat problem that contains m clauses over the set of variables $x_1, \ldots, x_n$, we start by making a small gadget of 3 vertices for each variable: one for the literal x_i, one for the literal $\neg x_i$, and a 3rd one. We connect those three vertices in a triangle.

We then add one more vertex to the graph for each clause, and put an edge between the clause's vertex and the (at most 3) literals it contains.

The result of our transformation is the resulting graph G and the parameter $k = n$.

If φ has a satisfying assignment, then the n vertices corresponding to the literals evaluating to true in this assignment form a dominating set of G: each vertex in a variable gadget is adjacent to the literal vertex in the set, and every clause vertex is adjacent to the literal vertex that satisfies the clause.

If φ does not have a satisfying assignment, then we claim G has no dominating set of size n. We prove the contra-positive statement: if G has a dominating set of size n, then φ has a satisfying assignment. Since each variable gadget has 1 vertex not connected to anything outside the gadget itself, any dominating set of G of size n must include exactly 1 vertex from each variable gadget. This vertex corresponds either to setting the variable to true, to false, or "don't care" (if the 3rd vertex is selected). So we can obtain an assignment to the variables from this dominating set. But since all the clause vertices must be adjacent to some vertex in the dominating set, it means that each clause is satisfied by at least one of the corresponding literals, so the assignment we obtained must be a satisfying assignment to φ.

3 3Sat and 3Col

In the 3Col problem, we are given as input a graph $G = (V, E)$ and we must determine if there is a way to colour the vertices in V using at most 3 colours so that each edge $(u, v) \in E$ connects vertices that have different colours. For this question, you will show that 3Col is **NP**-complete.

(i) Prove that 3Col $\in$ **NP**.

(ii) Prove that 3Sat $\leq_{\mathbf{P}}$ 3Col.

3.1 Solution

(i) We require as certificate the satisfying 3-colouring of the graph. A polynomial-time verifier can simply go through all the edges in E to verify that the 3-colouring is indeed valid. When G is 3-colourable, then there is at least one valid colouring that causes the verifier to return `Yes`. When G is not 3-colourable, no matter what assignment of colours to the vertices is provided to the verifier, one of the edges must connect two vertices with the same colour and the verifier always returns `No`.

(ii) Let φ be a 3SAT formula over n variables with m clauses. We want to construct a graph G using a similar technique to the previous question: we should have a gadget for each vertex and a gadget for each clause.

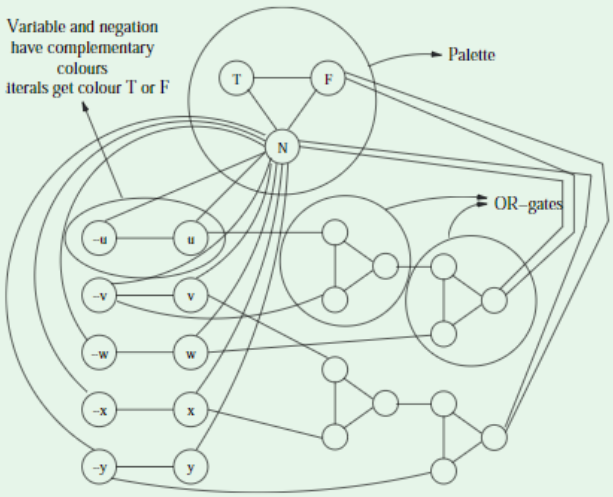
First, we will create a special 3-clique which establishes the meaning of the 3 possible colours. Label one node True, one False, and one Neutral. The idea is we will consider any vertex sharing a colour with True (False) to be logically true (false). The Neutral colour gives us some flexibility and allows us to specify vertices which must be either true or false.

Now we create a gadget for each variable consisting of two vertices labelled x_i and $\neg x_i$ with an edge between the two. Also, both of these vertices contain an edge to the Neutral vertex. This ensures that one of x_i and $\neg x_i$ is coloured True and one is coloured False.

Now we need to construct a gadget for each clause. Essentially, we want to construct an $\vee$-gate on 3 inputs and enforce that the output of the gate is coloured True. See the image below (borrowed from these slides) for how such a gadget would look.

Example

$\varphi = (u \vee \neg v \vee w) \wedge (v \vee x \vee \neg y)$



Suppose that φ has a satisfying assignment. Then we can construct a 3-colouring of G by colouring the gadgets for each literal accordingly, and then colouring the clause gadgets such

that the output nodes for the gadgets are all coloured True (you can verify that this is possible for all combinations of the input variables where at least one is True).

Suppose that G has a 3-colouring. Then, the colours applied to the gadgets for each variable yield a satisfying assignment for φ. Note that there are no contradictory literals since x_i and $\neg x_i$ cannot be the same colour and cannot be Neutral. Furthermore, the OR gate gadgets must have output gadget coloured True, and no such colouring is possible if both of the input literals are coloured False (you can verify this yourself by checking all possibilities).