CS3052 Poly Reduction Report

170011474

2020 May

1 Overview

This practical requires students to implement 3 reductions: from SAT to 3-SAT, from 3-SAT to K-Color problem, from K-Color problem to SAT. Additional extension is finished: from 3-SAT to vertex cover problem.

2 Usage

2.1 I/O Format

The input format for SAT is `Dimacs` CNF format, and the input format for graph-related problems is `Dimacs` graph format from http://prolland.free.fr/works/research/dsat/dimacs.html

```
c instructions
p edge 11 20
e 1 2
e 1 4
e 1 7
```

Please notice that for Graph problems, the output should be

```
c instructions
p edge 11 20
k 3
e 1 2
e 1 4
e 1 7
```

where the k is specified for Graph Coloring problem, Vertex Cover problems.

2.2 Running

To run the program, please specify input filename and output filename. `java SAT_TO_3SAT [input filename] [output filename]`

3 SAT $\leq_p$ 3-SAT

3.1 Algorithm

The algorithm is implemented based on Lecture 13 and Sanyal, n.d.. It iterates over each clause in the formula.

1. When a clause containing more than 3 literals. Splitting the clause into 2 shorter clause with a dummy literal added: $(w \lor x \lor y \lor z) = (d \lor w \lor x) \land (\neg d \lor y \lor z)$.
2. When the clause contains 2 literals, expands it into 2 clauses with one dummy literal added: $(x \lor y) = (d \lor x \lor y) \land (\neg d \lor x \lor y)$.
3. When containing 1 literal, expands it into 4 clauses with 2 dummy literals added: $(x) = (d_1 \lor d_2 \lor x) \land (\neg d_1 \lor d_2 \lor x) \land (d_1 \lor \neg d_2 \lor x) \land (\neg d_1 \lor \neg d_2 \lor x)$.

3.2 Polynomial Time Proof

The program is guaranteed to terminate because each split causes 2 or 4 shorter clauses. The reduction is polynomial because the number of splits needed is dependent on the length of the formula. So it is in $O(n)$. Besides, each split needs $O(n)$ to go over the length of the current clause. So altogether $O(n^2)$ which is in polynomial time.

3.3 Correctness Proof

Let us assume there is a formula $C = x_1 \lor x_2 \lor x_3 \lor x_4$ and converts it into $C_1 = (x_1 \lor x_2 \lor d) \land (\neg d \lor x_3 \lor x_4)$. Now we need to prove that the satisfiability is consistent. This means each split preserves the original satisfiability.

Firstly we show that a satisfying assignment for C leads to a satisfying assignment for C_1. If C is satisfiable, then at least one literal l is true. The literal l is either in the first clause in C_1 or in the second clause. If it is in the first clause, we can set $d = 0$ and thus $\neg d = 1$. So the second clause is also satisfiable as well the first clause. Therefore, C_1 is also satisfiable.

Secondly we show that a satisfying assignment for C_1 leads to a satisfying assignment for C. If C_1 is satisfiable, there are 2 possible cases: $d = 0$ or $d = 1$ in the first clause. If $d = 0$, then there is at least one literal l in the first clause is true. If $d = 1$, then there is at least one literal l in the second clause is true. So there is always a true literal except the dummy literal d, always resulting a satisfiable clause C.

Similarly, when the formula $C = x_1 \lor x_2$ and converts it into $C_1 = (d \lor x_1 \lor x_2) \land (\neg d \lor x_1 \lor x_2)$. If C is satisfiable, then there is at least one literal of x_1, x_2 is true, either in the first clause or the second one. We set the corresponding dummy literal as 1 in the clause without true literals so the C_1 can still be satisfiable. Conversely, similar to above. If $d = 0$, then there is at least one literal in the first clause is true. Otherwise, at least one in the second clause is true. There is always a true literal in C.

When the formula $C = x_1$ and converts it into 4 clauses. If C is satisfiable then x_1 must be true, resulting in 4 satisfiable clauses in C_1. Conversely, no matter what values d_1 and d_2 take in C_1, there must always be a true x_1, resulting in a satisfiable C (valid in this case).

3.4 Testing

We can manually construct an unsatisfied CNF formula. If we have n variables, then the unsatisfied formula for it has 2^n clauses. Those clauses are all possible combinations of variables. For example, if $n = 2$, then those 4 clauses are $(1 \vee 2) \wedge (1 \vee -2) \wedge (-1 \vee 2) \wedge (-1 \vee -2)$ and this formula must be unsatisfied. In this way, from any k-SAT formula to 3-SAT formula, we check if the 3-SAT instance as result of conversion is also unsatisfied. The SAT solver used here is at https://www.comp.nus.edu.sg/ gregory/sat/. After pressing the button, red color represents unsatisfied and green color means satisfied. To generate negative instances automatically, a `NegativeSATGenerator` is implemented in the source code.

In the implementation of the reduction, all negative instances is converted into negative instances in 3-SAT problems. In other words, if the instance before conversion is shown as red in the online solver, then the result of conversion is also shown as red in it.

All positive instances downloaded from `https://toughsat.appspot.com/` are turned into positive ones in 3-SAT. In other words, instances before conversion and results are all shown in green in the online solver. The typical positive examples have 1, 2, 4 literals within one clause. The reason why they are prove correctness is that they cover all edge cases.

4 3-SAT $\leq_p$ K-Color

4.1 Algorithm

The algorithm consists of 2 steps: the first one is to reduce 3-SAT to 3-Color, and the second one is to reduce 3-Color to K-Color Problem.

In the first step, each variable has a triangle and each clause has a gadget to connect. All triangles of variables have a common vertex of B (base vertex which preempts one color). T node for True node and F for False node.

In the second step, we would construct $k - 2$ base vertices to preempt $k - 2$ colors and leave 2 colors for true and false, similar to above. There is one base vertex for True and False triangle and **all other base vertices are connecting to every other nodes in the graph**.

4.2 Polynomial Time Proof

To construct a corresponding graph, the number of vertices $|V| = 2n + (k - 2) + 6m + 2$, where n is the number of variables, m is the number of clauses, $(k - 2)$ is the number of base

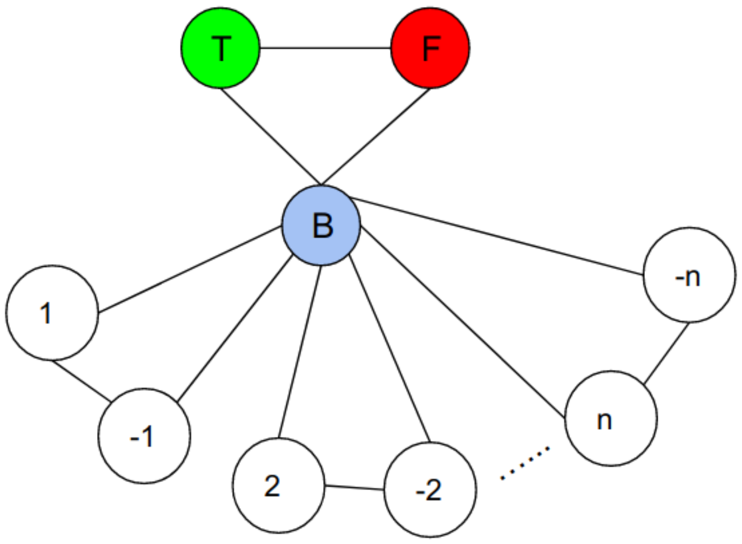


Figure 1: Connecting

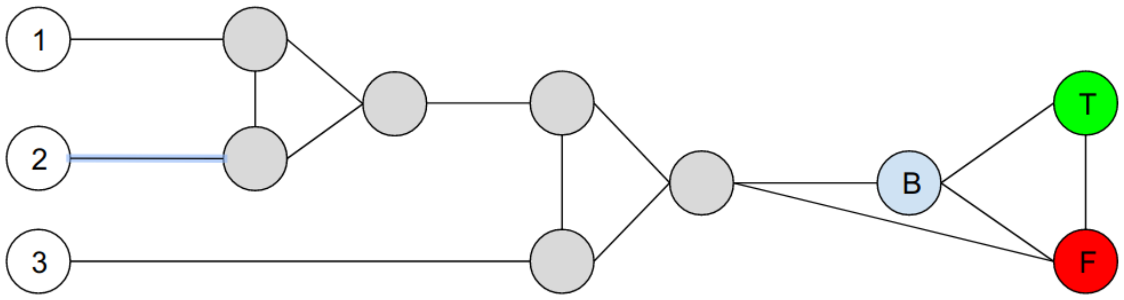


Figure 2: OR-Gadget

vertices, $6m$ is the number of gadget vertices with 6 vertices for each clause, and 2 for True and False. For example, if we have a variable x then there is also a negative variable $-x$, so $2n$ variable nodes and they takes $O(n)$. $6m$ vertices take $O(m)$. $k-2$ and 2 are constants taking $O(1)$. Therefore, constructing vertices is in $O(n)+O(m)$, in polynomial time.

The number of edges $|E| = n + \binom{k-2}{2} + 2n(k-2) + 2(k-2) + 6m(k-3) + 12m + 1$, where n is the number of edges between positive variables and negative variables, $\binom{k-2}{2}$ for connecting each base vertex with every other vertex, $2n(k-2)$ for connecting base vertices with variables and negative variables, $2(k-2)$ for connecting base vertices with True and False nodes, $6m(k-3)$ for connecting every base vertex with every gadget vertex except for one base node, $12m$ is the sum of the number of edges in gadgets, and 1 for edge between True and False.

From above, we can know $|E|$ is bounded. n for $O(n)$, $2n(k-2)$ for $O(n)$. For $\binom{k-2}{2}$, when you want $\binom{n}{k}$ from Pascal's triangle as a matrix, the complexity of building that matrix up to size nxk, since you can use $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$ to simplify it, then it would be $O(nk)$. In this case, $O((k-2)*2)$ so still constant $O(1)$. $6m(k-3)$ is bounded as $O(m)$. $12m$ for $O(m)$. Therefore, $2*O(n) + 2*O(m)$ so $O(n) + O(m)$.

Additionally, since NP is equivalent to PolyCheck, we can simply walk the graph and make certain that all adjacent vertices have a different colour, and make certain that only k colours are used. It is bound by $O(|V| + |E|)$.

In conclusion, the time complexity of constructing a graph is $2*(O(n)+O(c))$ so $O(n)+O(m)$.

4.3 Correctness Proof

The properties of the newly constructed graph are: in Figure 2, Figure 3 and Figure 4,

1. If a, b, c are all coloured F, then output node of OR-gadget has to be coloured F, which leads to unsatisfiability.
2. If one of a, b, c is coloured T, then there exists a valid 3-colorable graph.

4.3.1 Prove 3-SAT $\leq_p$ 3-Color

Firstly we prove that if 3-SAT instance ϕ is satisfiable then the graph G is 3-colorable. If ϕ is satisfiable and let$(x_1, x_2, ..., x_n)$ be the satisfying assignment. If x_i is assigned True, we color i with T and $-i$ with F. Since ϕ is satisfiable, every clause is satisfiable so the output node (the node connected to B in each gadget) is colored with T based on Property 2. The output node is connected with B and F with colour T, which leads to a proper colouring. Therefore, each clause with its gadget can be properly coloured and G is 3-colorable.

Secondly if G is 3-colorable, from Figure 4, we know the output node of each clause gadget must be coloured with T. If there is an unsatisfying assignment, which means there is at least one clause with all literals coloured with F. In this case, the output node must be coloured with F, which contradicts with the previous statement. Therefore, the 3-SAT instance ϕ must be satisfiable.

4.3.2 Prove 3-Color $\leq_p$ K-Color

By induction, we can show that G_k is k-colorable if and only if G_{k-1} is (k-1)-colorable.

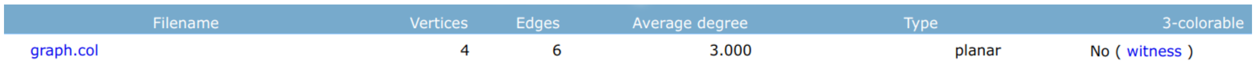
Assume G_{k-1} is (k-1)-colorable. Therefore, G_k is k-colorable because the added base vertex B_{k-3}, which is connected to every other vertices, can preempt k_{th} colour.

Assume G_k is (k)-colorable. Because the base vertex B_{k-3} is connected to all other vertices, B_{k-3} must be the only vertex with a certain colour. Therefore, other nodes are coloured with 1 of $(k-1)$ colours. Therefore, G_{k-1} is (k-1)-colorable.

By chaining this rule, we can conclude that 3-Color $\leq_p$ K-Color.

4.4 Testing

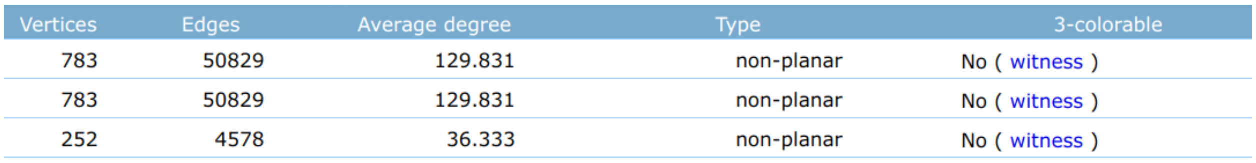
Since $3-Color \leq_p K-Color$ and also $k = 3$ can be considered from Karp's theorem, we can verify an instance's K-Colorability by verifying its' 3-Colorability. From an online solver for 3-color problem https://graph-coloring.appspot.com/, after uploading a `.col` file containing a graph, the result would show "no" if the graph cannot be colored in Figure 5:



Filename	Vertices	Edges	Average degree	Type	3-colorable
graph.col	4	6	3.000	planar	No (witness)

Figure 3: Online Solver for Graph Color

To test correctness, we can use unsatisfied SAT instances generated by `NegativeSATGenerator` and convert them into graphs. Theoretically, those graphs cannot be colored and we can verify them by the online solver.



Vertices	Edges	Average degree	Type	3-colorable
783	50829	129.831	non-planar	No (witness)
783	50829	129.831	non-planar	No (witness)
252	4578	36.333	non-planar	No (witness)

Figure 4: Negative Instances from 3-SAT to Graph Color when using n = 4, 5 in NegativeSATGenerator

Notice: When $n = 3$, the expected output should be "no" but the real output is "yes" in this online solver. I have debugged it and found that the given solution provided by it requires vertices "1, 2, 3, 4" to be coloured with the same colour "1" while in the `graph.col` there is a line `e 1 4`. In this case, 2 endpoints are all coloured with the same so this solver is wrong here.

Additional testings are conducted by multiple simple graphs. After testing $K - Color \leq_p SAT$, I have tried to do reductions from $3 - SAT \leq_p K - Color \leq_p SAT$ and check the 3-SAT and SAT results. Their satisfiability values are all the same. Positive instances picked are all within 5 variables. The reason for this is that it shows that it can handle proper new vertices outside 3 and allocate new coloured vertices and make correct gadgets.

5 K-Color $\leq_p$ SAT

5.1 Algorithm

According to Dey and Bagchi in 2013,a graph with k color as a constraint can be converted to 3-SAT problems through 3 types of clauses. Type 1 clauses ensure that 2 adjacent nodes do not share the same color. Type 2 clauses ensure that each node is assigned at least one color. Type 3 clauses ensure that each node is assigned at most one color. Each node would be represented by a number of nodes and the number is the number of color k.

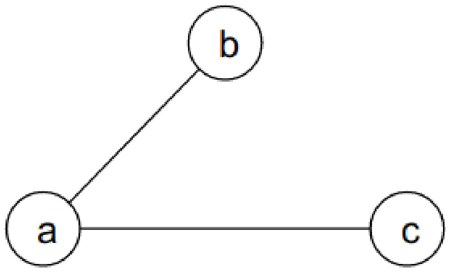


Figure 5: Example of graph with 3 colours

For example, in Figure 5, we have 3 colours, then for each node we use 3 variables to represent 3 different colours:

1. Type 1: $\neg a_1 \vee \neg c_1$, $\neg a_2 \vee \neg c_2$, $\neg a_3 \vee \neg c_3$ for edge (a, c)
2. Type 2: $a_1 \vee a_2 \vee a_3$ for vertex a.
3. Type 3: $\neg a_1 \vee \neg a_2$, $\neg a_1 \vee \neg a_3$, $\neg a_2 \vee \neg a_3$ for vertex a.

All 3 types of clauses are some constraints. Type 1 clauses belong to **edge constraint** and type 2 and type 3 clauses belong to **vertex constraints**. In the following notations, v_i means type 1 and type 2 clauses for vertex i.

5.2 Polynomial Time Proof

Assume that the number of nodes is denoted by n, and the number of colors is denoted by k.

From the paper, the number of type 1 clauses is number of nodes in the graph multiplied by the number of colors, which is $O(nk)$. The number of type 2 clauses is number of nodes in the graph, which is $O(n)$. The number of type 3 clauses is number of nodes in the graph multiplied by the number of color, which is $O(nk)$. The number of new variables needed is number of nodes multiplied by the number of color, which is $O(nk)$. All together, the time is $O(nk) + O(n) + O(nk) + O(nk) = 3 * O(nk) + O(n)$. Ignoring constants and considering the dominants, then the running time is $O(nk) + O(n)$.

5.3 Correctness Proof

Assume that an undirected graph $G(V, E)$ that is k-colorable and the following is a SAT formula corresponding to the graph G:

$$F = (v_i \wedge e_j)$$

where $v_i = v_1, v_2, ..., v_n$ are n vertices and $e_j = e_1, e_2, ..., e_m$ are m edges. The formula F can be expanded as

$$F = F_v \wedge F_e = ((v_1 \wedge v_2 \wedge ... \wedge v_n) \wedge (e_1 \wedge e_2 \wedge ... \wedge e_m))$$

where F_v is the $(v_1 \wedge v_2 \wedge ... \wedge v_n)$ and F_e is the $(e_1 \wedge e_2 \wedge ... \wedge e_m)$.

Firstly we need to show that if the graph is k-colorable, then the SAT formula is satisfying. This is obvious, because if the graph is k-colorable, it means that 2 endpoints on an edge do not share the same color, which satisfies each e_j clause. Besides, it also means that every vertex has at most one color and at least one color, which gives type 2 clauses and type 3 clauses for every vertex v_i a satisfying assignment. Therefore, every clause in the formula F is satisfying and so is F.

Secondly we need to show that if formula F is satisfiable, then the graph is k-colorable. If F is satisfiable, then F_v is satisfiable and F_e is satisfiable. In this case, every v_i is satisfied so every vertex has exactly one color. Similarly, every e_j is satisfied so every edge has 2 differently colored endpoints. Therefore, the graph is k-colorable.

5.4 Testing

Based on successful reductions provided above (SAT $\leq_p$ 3-SAT and 3-SAT $\leq_p$ K-Color), we can use `NegativeSATGenerator` to generate negative instances and reduce them to negative instances of K-Color graphs. Afterwards, those graphs are converted into SAT instances again and we check if they are all still negative.

By the online SAT solver with link above, all negatives instances of graphs are converted to negative SAT instances by showing red colours. Typical positive instances all picked among instances within 5 variables. Since for a graph colouring problem, the problem maintains a homogeneous essence: representing one vertex with k vertices in different colours and then construct 3 types of clauses. Those problems are sufficient to show the procedure.

6 Extension: 3-SAT $\leq_p$ Vertex Cover

6.1 Algorithm

It takes 2 steps, 3-SAT $\leq_p$ Clique, and Clique $\leq_p$ Vertex Cover. They are all derived from Introduction to Algorithm. In the first part, if we have a 3-SAT instance $\phi = C_1 \wedge C_2 \wedge ... \wedge C_k$. For each clause C_r, we construct a triple with at most 3 nodes to represent literals in the clause. Edges are placed between 2 nodes if and only if the 2 nodes:

1. They are in different triples (clauses).
2. They are not negation of each other.

In the second part, simply constructs a complement graph G' of the graph G in Clique problem, which is the graph containing exactly those edges that are not in G.

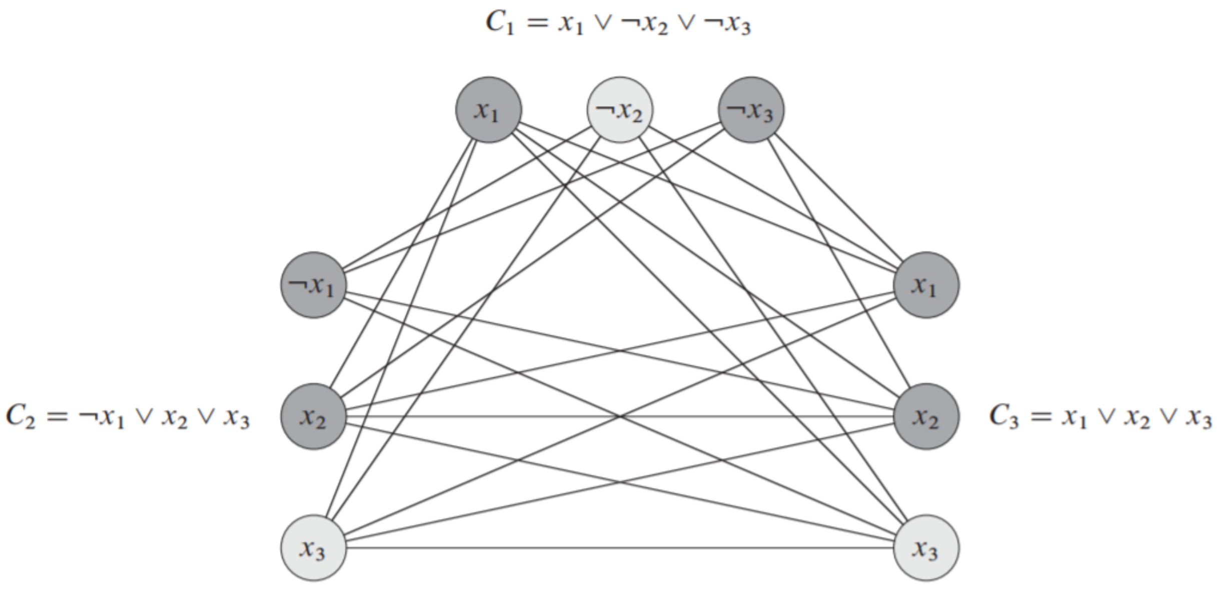

$C_1 = x_1 \vee \neg x_2 \vee \neg x_3$
$C_2 = \neg x_1 \vee x_2 \vee x_3$
$C_3 = x_1 \vee x_2 \vee x_3$
x_1
$\neg x_2$
$\neg x_3$
$\neg x_1$
x_2
x_3
x_1
x_2
x_3

Figure 6:

6.2 Polynomial Time Proof

6.2.1 Clique

For a given graph $G = (V, E)$ where V is the collection of all vertices and E is the collection of all edges. Let the set $V' \subseteq V$ in the clique as a proof of the problem. We can check whether V' is a clique in polynomial time by verifying if, for each pair $u, v \in V'$, the edge $(u, v) \in E$. The time required is bounded to the number of vertices except the current triple's vertices. So Clique $\in$ NP.

To convert 3-SAT to Clique problem, creating vertices requires $O(n)$ where n is the number of literals altogether. The number of edges is bounded by the number of vertices available, so $O(n)$ to iterate each vertex and $O(n)$ for connecting edges. $O(n) + O(n^2)$ so $O(n^2)$.

6.2.2 Vertex Cover

Given a graph $G = (V, E)$ and an integer k. Let the set $V' \subseteq V$ be a proof of the problem. After checking $|V'| = k$, it checks for each edge (u, v) in E, if $u \in V'$ or $v \in V'$. If so, remove all edges adjacent to the vertex. When there is no left edge, check if the size if k. So Vertex Cover $\in$ NP.

To convert Clique problem with size k into Vertex Cover problem, we only need to construct the complement graph G'. There is no need to create new vertices and iterate over each vertex, we also check if there are other vertices not connected with it in the original graph G. So $O(n^2)$.

6.3 Correctness Proof

6.3.1 3-SAT $\leq_p$ Clique

Firstly, we can show that if ϕ has a satisfying assignment, then graph G has a clique with size k, where k is the number of clauses. Assume that ϕ has a satisfying assignment. Each clause C_r contains at least one true literal, and each such literal corresponds to a vertex v_i^r. If we pick one such true literal vertex from every clause, then a set V' of k vertices come up. This set V' is a clique with size k, because we already know the rules of constructing graph G is that 2 endpoints of an edge cannot be in the same clause. Since they are all mapped to true, and thus the literals are not complements. Therefore, every edge $(v, u) \in E$ where $v, u \in V'$.

Secondly, we can show that if graph G has a clique with size k, then ϕ has a satisfying assignment. Assume that graph G has a clique V' with size k. Since there is no edge in G connect vertices in the same triple, V' contains exactly 1 vertex in each triple. By assigning true to each literal vertex $v \in V'$ without fear of assigning true to both a literal and its negation, since G has no edges connecting complement literal vertices. In this way, each clause (triple) is satisfied and so ϕ is satisfied.

6.3.2 Clique $\leq_p$ Vertex Cover

Firstly, assume that there is a clique of size k in G. Let the set of vertices in the clique be V'. This means $|V'| = k$. In the complement graph G' for Vertex Cover problem, pick any edge (u, v). Then at least one of u or v must be in the set $V–V'$. This is because, if both u and v were from the set V', then the edge (u, v) would belong to V', which means that the edge (u, v) is in G. This contradicts to the fact that (u, v) is not in G. Thus, all edges in G' are covered by vertices in the set $V–V'$.

Secondly, assume that there is a vertex cover V'' of size $|V|–k$ in G'. This means that all edges in G' are connected to some vertex in V''. As a result, if we pick any edge (u, v) from G', both of them cannot be outside the set V''. This means, all edges (u, v) where both u and v are outside the set V'' are in G, i.e., these edges constitute a clique of size k.

6.4 Testing

The brute force solver is from `https://github.com/pranav91/vertexCover` and the source code has been adapted to a version which can read input file. Negative instances converted from $n = 1$, $n = 2$, $n = 3$ from `NegativeSATGenerator` have been checked. For a larger n, the brute force solver requires exponentially larger time so they are not checked. Positive 3-SAT instances where the number of variables controlled within 5 are converted into positive Vertex Cover instances.

All outputs have been reviewed based on the algorithm. Since the number of vertices should be $|V| = 3m$ where m is the number of clauses, and number of edges should be $|E| = 2^n - m$ where n is the number of variables. The k value of the vertex cover problem is $|V| - m$. This numeric checks are also conducted and are all satisfied.

The reason why those experiments prove the reduction is correct is that they are typical enough. The number of variables is 5 is typical because it shows that it can handle proper new vertices outside 3 and get the original literal values for vertices to compare. The numeric check for resulting $|V|$ and $|E|$ shows that correct size of converted problem is made and the algorithm works properly.

References

[1] Dey, M. and Bagchi, A., 2013. *Satisfiability Methods For Colouring Graphs.* [online] https://www.researchgate.net. Available at: https://www.researchgate.net/publication/268460124_Satisfiability_Methods_for_Colouring_Graphs/fulltext/5721424d08ae0926eb45bd3f/Satisfiability-Methods-for-Colouring-Graphs.pdf [Accessed 23 April 2020].

[2] pranav91. 2016. *Vertexcover.* [online] Available at: https://github.com/pranav91/vertexCover [Accessed 7 May 2020].

[3] Sanyal, S., n.d. *Reduction From SAT To 3SAT.* [online] Cse.iitkgp.ac.in. Available at: https://cse.iitkgp.ac.in/~palash/2018AlgoDesignAnalysis/SAT-3SAT.pdf [Accessed 6 May 2020].

[4] Sharma, P. and Chaudhari, N., 2012. *A New Reduction From 3-SAT To Graph Kcolourability For Frequency Assignment Problem.* [online] Pdfs.semanticscholar.org. Available at: https://pdfs.semanticscholar.org/1b7d/e31398887fe79f5b363a62ab493672027293.pdf [Accessed 3 May 2020].