Reduction from 3SAT to 3COLORING

Xiaoqi's Project Report, GTIIT

January 13, 2025

Abstract

This report introduces the polynomial-time reduction with example. We firstly use a non-deterministic WHILE language program to show 3COLORING ∈ NP.Then by assuming 3SAT ∈ NPC, we conducted a full reduction from 3SAT to 3COLORING with detailed explanation, and wrote a formal Pseudocode algorithm for the reduction. In this way, we proved 3SAT $\preceq$ 3COLORING, and 3COLORING ∈ NPC.

1 Introduction

1.1 Definitions

Definition 1. *Let $A \subseteq D_{01}$. It follows that $A \in$ NP iff there is a polynomial time non-deterministic WHILE program P s.t. $a \in A$ iff $(a,$ true$) \in [P]$.*

We would like identify what are the *hardest problems* in NP in some precise sense. The idea is to empoly a technique to compare problems regarding their difficulty called *reduction*.

Definition 2. *The hardest problems in NP give rise to a new complexity class, called NPC.*

Definition 3. *Let A and B be two decision problems, and B be a solver for B. A polynomial-time reduction from A to B, written $A \preceq B$, is a polynomial-time program P_B s.t. $a \in A$ iff $(a,$ true$) \in [P_B]$.*

In words, a polynomial-time reduction states the existence of an program which, using the program for solving B, builds in polynomial time a solver for A.

Definition 4. *A decision problem A is NP-complete iff*

1. *$A \in$ NP.*
2. *For any other problem $B \in$ NP, we have $B \preceq A$.*

Theorem 1. *Given decision problems $A \in$ NP and $B \in$ NPC, to prove $A \in$ NPC, we can prove $B \preceq A$.*

Proof. $B \in$ NPC implies for any other problem $C \in$ NP, we have $C \preceq B$. Since we also have $B \preceq A$ by assumption, and $\preceq$ is transitive, for any other problem $C \in$ NP, we have $C \preceq A$. □

1.2 3SAT

A boolean formula is in *conjunctive normal form* (CNF) if it is a conjunction (AND) of several clauses, each of which is the disjunction (OR) of several *literals*, each of which is either a variable or its negation.

A special case of SAT is called 3CNF-SAT or more often simply 3SAT. A 3CNF formula is a CNF formula with exactly 3 literals per clause. 3SAT is the restriction of SAT to 3CNF formulars: Given a 3CNF formula, is there an assignment to the variables that makes the formula evaluate to TRUE?

For example:

$$(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d)$$

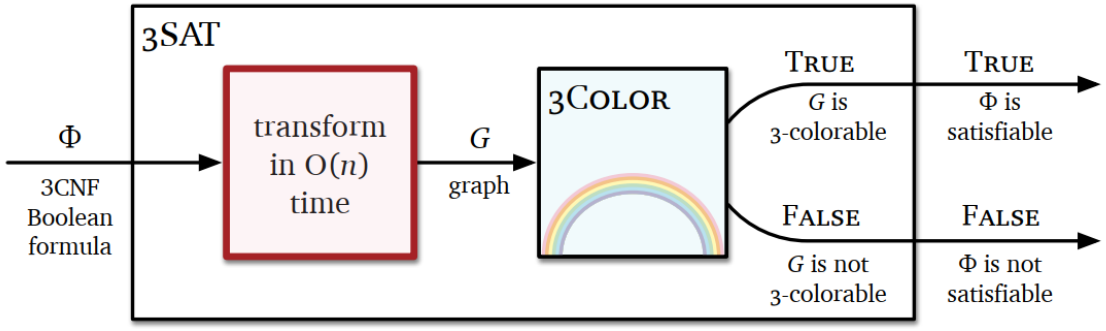
1.3 3COLRONG

A *proper k-coloring* of a graph $G = (V, E)$ is a function $C : V \longrightarrow \{1, 2, ..., k\}$ that assigns one of k colors to each vertex, so that every edge has two different colors at its endpoints.

The graph coloring problem asks for the smallest possible number of colors in a legal coloring of a given graph. And a special case of `COLORING` is called `3COLORING`: Given a graph, does it have a proper 3-coloring?

1.4 Goal

Our goal for this report is to prove `3COLORING` ∈ `NPC`, in this case, we will assume `3SAT` ∈ `NPC` and show `3SAT` ⪯ `3COLORING` by giving a `WHILE` program $P_{\texttt{3COLORING}}$, s.t. ∀ `3CNF` formula Φ, it can determine whether it's satisfiable by evaluating whether the corresponding graph is 3-colorable.



2 Non-deterministic `WHILE` program to prove `3COLORING` ∈ `NP`

Firstly, following Definition 1, we use a non-deterministic `WHILE` program to show `3COLORING` ∈ `NP`:

```
/* Find Color helper function */
read X;
    Y := nil;
    GO := true;
    cur_node := hd X;
    Nodes := tl X;

    while GO do
        rewrite Nodes by
            [(N, C), Rest] =>
                if N =? cur_node do
                    Y := C;
                    GO := false
                else
                    Nodes := Rest;
            [nil] => GO := false;
write Y;

/* Eval helper function - Validates color assignments */
read X;
    Y := true;
    Nodes := hd X;
    Edges := tl X;
    GO := true;
```

```
25
26      while GO do
27          rewrite [Edges] by
28              [nil] => GO := false;
29              [(N1, N2), Rest] =>
30                  Color_N1 := find_color(N1, Nodes);
31                  Color_N2 := find_color(N2, Nodes);
32                  if Color_N1 =? Color_N2 then
33                      Y := false;
34                      GO := false
35                  else
36                      Nodes := Rest;
37  write Y;
38
39  /* 3-Coloring Program - Main program that attempts to find a valid 3-coloring */
40  read X;
41      N := hd X;
42      Edges := tl X;
43      Nodes := [];
44
45      while N do
46          C := choose [0, 1, 2];  /* Non-deterministically choose color 0, 1, or 2 */
47          N := N-1;
48          Nodes := Nodes # [N, C];  /* Append node-color pair to list */
49
50      Y := Eval(Nodes cons Edges);
51  write Y;
52
53
54  /* Examples of nested choose statements */
55  C := choose 0 1;
56  D := choose C 2;
57  /* Alternative nested choose syntax */
58  C := choose [0, choose [1, 2]];
59
```

3 Reduction from formula Φ to graph G

Now, we know 3COLORING $\in$ NP, we can assume 3SAT $\in$ NPC and start the reduction.

3.1 Decomposing graph into *gadgets*

The formula-to-graph reduction uses three types of gadgets:

- *Truth gadget*: There is a triangle with three vertices T, F, and X, which stand for `True, False`, and `Other`, also for 3 colors. Namely, when we say that a vertex is colored `True`, we mean that it has the same color as vertex T.
- *Variable gadget*: For each variable a, there is a triangle joining two new vertices labeled a and $\bar{a}$ to the vertex X in the truth gadget. Vertex a must be colored either `True` or `False`, while $\bar{a}$ will be colored in opposite way.
- *Clause gadget*: For each clause $(a \vee b \vee \bar{c})$ in Φ, there is a new gadget using five new unlabeled nodes, as shown below.

Remark 1. Rules to color *Clause gadget*:

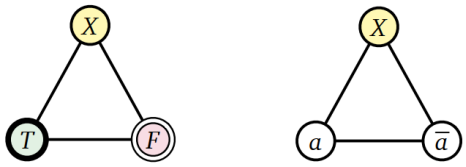


Figure 12.12. The truth gadget and a variable gadget for a.

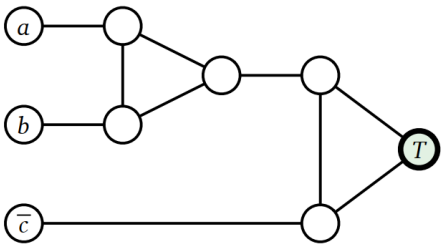


Figure 12.13. A clause gadget for $(a \vee b \vee \bar{c})$.

1. We will divide a Clause gadget into 2 "half-gadget", upper and lower half-gadets.
2. For each upper half-gadget, if the two vertices on the left have the same color, the rightmost vertex must have the same color, otherwise, the color of the rightmost vertex can be chosen arbitrarily.

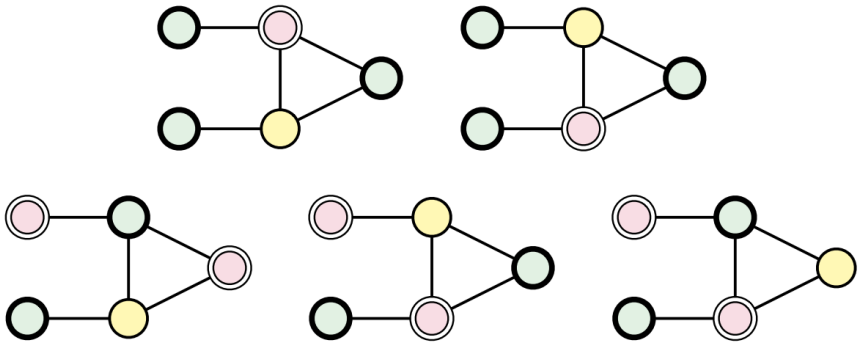


Figure 12.14. All valid 3-colorings of a "half-gadget", up to permutations of the colors

3. Notice that for each lower half-gadget, the rightmost vertex must be vertex T.
4. Notice that there is no valid 3-coloring where three literal nodes of a clause are colored `False`. In other words, in any valid 3-coloring of the clause gadget, at least one literal node is colored `True`.

4 Algorithm with example

Take the example: $\Phi = (a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$. It is satisfiable with the assignmnet $a = c =$ `True`, and $b = d =$ `False`. We will show the corresponding graph is also 3-colorable.

Procedures 1. Reduce Φ to G:

1. If we denote n as the number of variables, m as the number of clauses in Φ, the constructed graph G will have $2 \cdot n + 5 \cdot m + 3$ vertices and $3 \cdot n + 10 \cdot m + 3$ edges. In this case, we will construct G with 31 vertices and 55 edges.

2. If we denote the vertices of a Clause gadget to be $i0, i1, i2, i3, i4$, where i is the index of Clause gadget, Figure 1. Then 31 vertices are: X, T, F, a, $\bar{a}$, b, $\bar{b}$, c, $\bar{c}$, d, $\bar{d}$, 00, 01, 02, 03, 04, 10, 11, 12, 13, 14, 20, 21, 22, 23, 24, 30, 31, 32, 33, 34, 40, 41, 42, 43, 44.

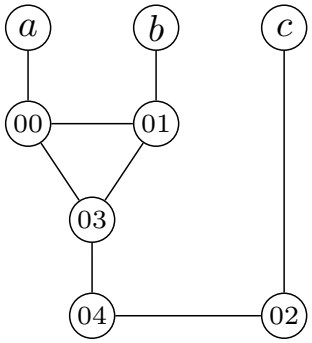


Figure 1:

3. We put the edges here and connect the edges within *truth gadget*, *variable gadgets*, and *Clause gadgets*. We color `True` as green, `False` as red, and `Other` as yellow, Figure 2.

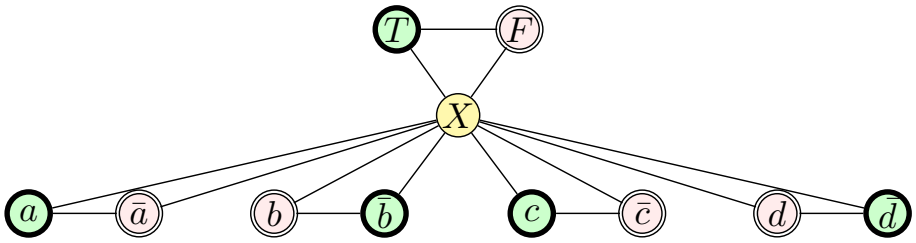


Figure 2:

4. For each clause, we connect the corresponding variables with $i0, i1, i2$.

5. Then we start coloring the Clause gadget following the rules in Remark 1.

   (a) Clause gadget 1:

      i. Since a is green and b is red, 03 can choose an aribitrary color, here is yellow, Figure 3. Then 00 can only be colored by red and 01 by green, Figure 4.

      ii. Since 04 is connected with T, which is green, meanwhile 03 is yellow, 04 can only be red, Figure 4.

      iii. Now the only left vertex is 02, which is connected with c(green), 04(red), and T(green), thus 02 can only be colored by yellow, Figure 5.

      iv. In this way, the first clause is 3-colorable.

   (b) Clause gadget 2:

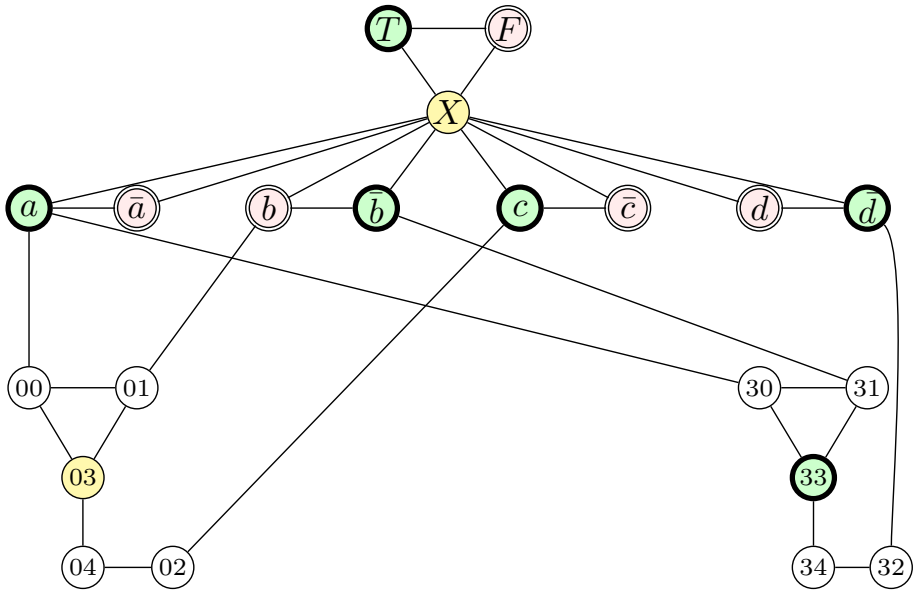


Figure 3:

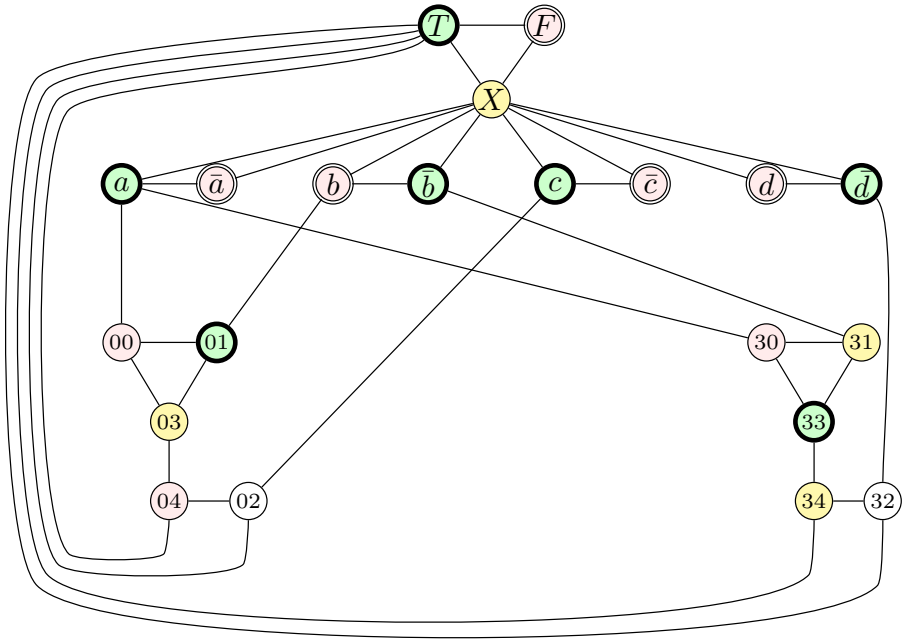


Figure 4:

i. Since b is red and $\bar{c}$ is red, 13 must have the same color, red, then 10 is colored by green arbitrarily, and 11 can only colored by yellow.

ii. Since 14 is connected with T, which is green, meanwhile 13 is red, 14 can only be yellow.

iii. Now the only left vertex is 12, which is connected with $\bar{d}$(green), 14(yellow), and T(green), thus 12 can only be colored by red.

iv. In this way, the second clause is 3-colorable.

(c) Clause gadget 3:

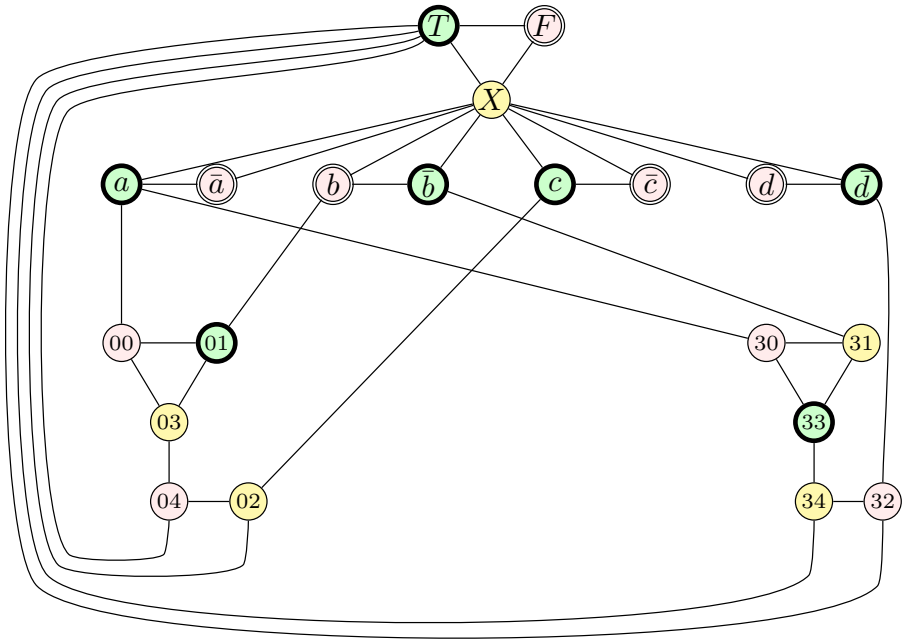


Figure 5:

i. Since $\bar{a}$ is red and c is green, 23 can choose an aribitrary color, here is green, then 20 can only be colored by yellow and 21 by red.

ii. Since 24 is connected with T, which is green, meanwhile 23 is green, 24 can choose an arbitrary color, here is red.

iii. Now the only left vertex is 22, which is connected with d(red), 24(red), and T(green), thus 22 can only be colored by yellow.

iv. In this way, the third clause is 3-colorable.

(d) Clause gadget 4:

i. Since a is green and $\bar{b}$ is green, 23 must have the same color, green, then 30 is colored by red arbitrarily, and 31 can only colored by yellow.

ii. Since 34 is connected with T, which is green, meanwhile 33 is green, 34 can choose an arbitrary color, here is yellow.

iii. Now the only left vertex is 32, which is connected with $\bar{d}$(green), 34(yellow), and T(green), thus 32 can only be colored by red.

iv. In this way, the fourth clause is 3-colorable.

6. Since all Clause gadgets are 3-colorable, the graph G is 3-colorable, Figure 6, thus the formula Φ is satisfiable.

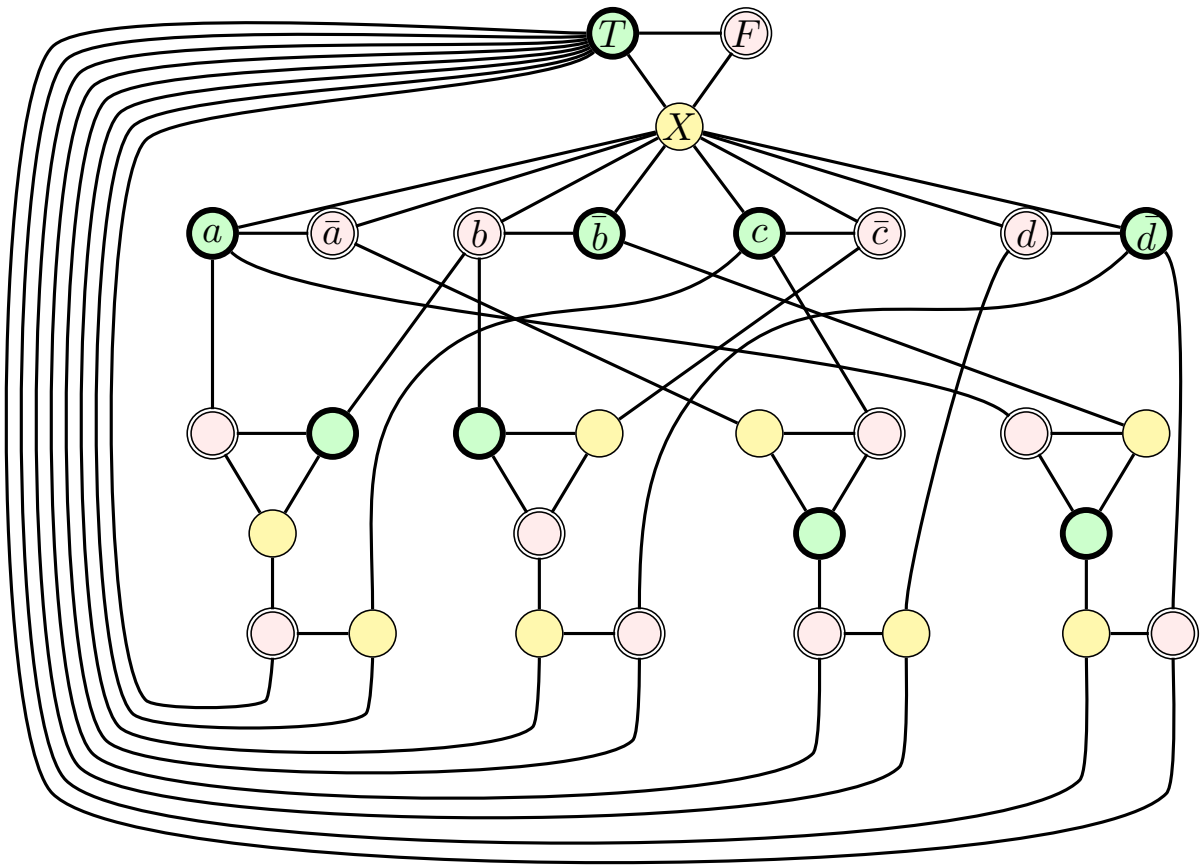


Figure 6: $(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$

5 Pseudocode for the algorithm

Algorithm 1 Reduction from Φ to G

Input: A 3CNF formula $\Phi = \wedge(cf_i), cf_i = (y_{i0} \vee y_{i1} \vee y_{i2})$ with n variables x_i and m clauses.
Output: Φ is satisfiable or not.

procedure REDUCTION($\Phi = \wedge(cf_i), n, x_i, m$)
  $G \leftarrow$ GRAPH(Φ, x_i, n, m) ▷ construct graph of Φ using Algorithm 2
  $\{color[T], color[F], color[X]\} \leftarrow \{0, 1, 2\}$ ▷ denote colors of T, F, X
  $i \leftarrow 0$
  while $i < n$ **do** ▷ assign colors to variables in a non-deterministic way
    $color[x_i] \leftarrow$ choose $\{0, 1\}$ ▷ either $True$ or $False$
    $color[\bar{x_i}] \leftarrow$ choose $\{0, 1\} \setminus \{color[x_i]\}$
    $i \leftarrow i + 1$
  end while
  $i \leftarrow 0$
  while $i < m$ **do** ▷ color Clause gadgets
    if $color[y_{i0}] = color[y_{i1}]$ **then** ▷ color $i3$
      $color[i_3] \leftarrow color[y_{i0}]$
    else
      $color[i_3] \leftarrow$ choose $\{0, 1, 2\}$
    end if
    $color[i0] \leftarrow$ choose $\{0, 1, 2\} \setminus \{color[y_{i0}], color[i3]\}$ ▷ color $i0$
    $available\_colors \leftarrow \{0, 1, 2\} \setminus \{color[y_{i1}], color[i0], color[i3]\}$ ▷ color $i1$
    if $available\_colors = \emptyset$ **then**
      return False
    end if
    $color[i1] \leftarrow$ choose $available\_colors$
    $color[i4] \leftarrow$ choose $\{0, 1, 2\} \setminus \{color[i3], color[T]\}$ ▷ color $i4$
    $available\_colors \leftarrow \{0, 1, 2\} \setminus \{color[y_{i2}], color[i4], color[T]\}$ ▷ color $i2$
    if $available\_colors = \emptyset$ **then**
      return False
    end if
    $color[i2] \leftarrow$ choose $available\_colors$
    $i \leftarrow i + 1$
  end while
  return True
end procedure

Algorithm 2 Construct gragh G from Φ

Input: $\Phi = \wedge(cf_i), cf_i = (y_{i0} \vee y_{i1} \vee y_{i2}), x_i$, n is the number of variables, m is the number of clauses.
Output: An undirected graph $G = (V, E)$ with $2 \cdot n + 5 \cdot m + 3$ vertices and $3 \cdot n + 10 \cdot m + 3$ edges.

procedure GRAPH($\Phi = \wedge(cf_i), n, m$)
  $G \leftarrow \{V, E\}$
  $V \leftarrow \{X, T, F\}$
  $E \leftarrow \{(X, T), (T, X), (T, F)\}$
  $i \leftarrow 0$
  while $i < n$ **do**
    $V \leftarrow V \cup \{x_i, \bar{x_i}\}$ ▷ x_i's are the variables in Φ
    $E \leftarrow E \cup \{(x_i, X), (\bar{x_i}, X), (x_i, \bar{x_i})\}$
    $i \leftarrow i + 1$
  end while
  $i \leftarrow 0$
  while $i < m$ **do**
    $V \leftarrow V \cup \{i0, i1, i2, i3, i4\}$ ▷ vertices of Clause gadgets
    $E \leftarrow E \cup \{(i0, i1), (i0, i3), (i1, i3), (i3, i4), (i2, i4), (i4, T), (i2, T)\}$ ▷ edges of Clause gadgets
    $j \leftarrow 0$
    while $j < 3$ **do**
      $E \leftarrow E \cup \{(y_{ij}, ij)\}$ ▷ edges from variables to gadgets
      $j \leftarrow j + 1$
    end while
    $i \leftarrow i + 1$
  end while
  return G
end procedure

6 Conclusion

`3COLORING` $\in$ `NPC`

References

[1] LJones, Neil D. Computability and complexity: from a programming perspective. *MIT press*, 1997

[2] J Erickson. Algorithms. *ISBN: 978-1-792-64483-2*, 2019

[3] V Cassano, Lectures of *Theory of Computation*, GTIIT