Graph Algorithm: Home assignment

Louis Milhaud

November 2023

1 Exact exponential algorithms for the Graph Colouring Problem

In every question that mentions colourating graphs, we can assume that colouring the disconnected parts of a graph is 1-colourable: just assign to all disconnected vertices the first colour, so it won't be specified.

Question 1.1. *To begin, let us find a first non-trivial algorithm for 3-COLOUR.*

(a) Given a tree T of size n, lets calculate the number of proper 3-colouring we can find. One can calulate it with the following:
Let K be the set of proper colourings; $\forall 1 \leq i \leq n, v_i \in V(T)$; ϕ the colouring map and $\Phi_{v_i} := \{\phi(v_i) | \phi$ is a proper colouring$\}$

$$|K| = \prod_{i=1}^{n} |\Phi_{v_i}| \qquad *$$

- *Basic Case:*
Let's choose $v_r \in V(T)$ a vertex, it will be the root. Therefore $\Phi_{v_r} = \{1, 2, 3\}$ and $|\Phi_{v_r}| = 3$.
To colour properly the rest of the tree, we apply the *Inductive Case* recursively to the children. Starting with the children of v_r.
- *Inductive Case:*
Let v_k be a vertex, by the induction hypothesis v_k has a parent $v_j \in N_T(v_k)$ and $\phi(v_j) = c, \quad c \in \{1, 2, 3\}$. Also $\forall v_l \in N(v_k)$ such that $v_l \neq v_j$, we claim v_l isn't coloured yet. If it was the case, then by induction it means that v_l is a parent of v_k, thus there exist a common vertex, parent of v_k and v_l, thus there exist a cycle in T. Which is absurd because T is a tree.
So $\Phi_{v_k} = \{1, 2, 3\} \backslash \{c\}$ and $|\Phi_{v_k}| = 2$.

We then apply the formula $*$:

$$\begin{aligned}|K| &= \prod_{i=1}^{n} |\Phi_{v_i}| \\ &= |\Phi_{v_r}| \cdot \prod_{i=2}^{n} |\Phi_{v_i}| \\ &= 3 \cdot \prod_{i=1}^{r-1} |\Phi_{v_i}| \cdot \prod_{i=r+1}^{n} |\Phi_{v_i}| \\ &= 3 \cdot 2^{n-1}\end{aligned}$$

(b) Let's derive from above an algorithm that solves 3-COLOUR:

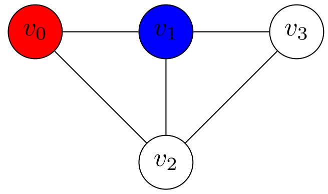
Algorithm 1 $O^*(2^n)$ algorithm to solve 3-COL problem

```
C ← {1, 2, 3}
function colourChildren(v: vertex, ST: tree, φ: map(vertex, int))
    for each w ∈ children(v) do
        c_wasted ← {φ(v)}
        repeat
            if c_wasted = C then
                return false
            end if
            c ← pickColour(C\c_wasted)
            φ[w] ← c
            c_wasted.add(c)
        until colourChildren(w, ST, φ)
    end for
    return true
end function
ST ← spanningTree(G)
v ← pickNode(V(ST))
φ ← {v → pickColour(C)}
colourChildren(v, ST, φ)
```

Question 1.2. *Given a graph G, a dominating set of G is a set of vertices $X \subseteq V(G)$ such that $N_G[X] = V(G)$.*

(a) To extend the colouring to the entire graph in polynomial time we will build a polynomial reduction from this problem to 2-SAT.
Intuition:

- Let $G = (\{v_0, v_1, v_2, v_3\}, \{v_0v_1, v_0v_2, v_1v_2, v_1v_3, v_2v_3\})$
- Let $X = \{v_0, v_1\}$

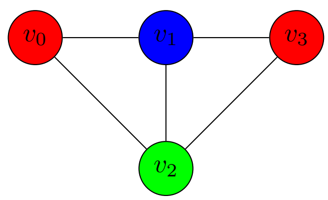


The reduction encodes the following clauses C_2, $C_{3,a}$, $C_{3,b}$, $C_{2,3,g1}$ and $C_{2,3,g2}$ corresponding to v_2, v_3:

- Let r_i, g_i, b_i the possible variables for clause i
- Variables r_2, b_2, b_3 aren't in the clauses because of X (proper colouring conditions)
- $C_2 = (g_2)$ $\quad C_{3,a} = (g_3 \vee r_3)$ $\quad C_{3,b} = (\neg g_3 \vee \neg r_3)$ $\quad C_{2,3,g1} = (g_2 \vee g_3)$ $\quad C_{2,3,g2} = (\neg g_2 \vee \neg g_3)$

The only solution to the 2-SAT problem is:

$$\{g_2\text{: true, } g_3\text{: false, } r_3\text{: true, } b_3\text{: false}\}$$



<u>Reduction:</u>
We define the following:

- Let X a coloured dominating set of G
- Let r, g, b be the 3 different colours
- Let ϕ the colouring map
- $\forall v \in V(G)\backslash X$, let $r_v, g_v, b_v \in \mathbb{B}$ the colouring variables
- $\forall v \in V(G)\backslash X$, let $C_{v,a}$ and $C_{v,b}$ the 2-clauses corresponding to v and the dominating neighbours
- $\forall v, w \in V(G)\backslash X$ such that $vw \in E(G)$, let $C_{v,w,c1}$ and $C_{v,w,c2}$ with c a colour, the 2-clauses for non dominating neighbors

We have the following properties:

1. $\forall v \in X \ \phi(v) \in \{r, g, b\}$
2. Let $x \in X$, $\forall v \in N_G(x)$ the proper colouring condition imposes:

$$C_{v,\_} \text{ must not contain } \phi(x)$$

To ensure the reduction we need to build $\forall v \in G$ their corresponding clauses in a way that finding a valid solution to the 2-SAT problem:

$$\bigwedge_{v \in V(G)\backslash X} (C_{v,1} \wedge C_{v,2} \bigwedge_{w \in N_G(v)\backslash X} (C_{v,w,c1} \wedge C_{v,w,c2} \wedge C_{v,w,d1} \wedge C_{v,w,d2}))$$
$$\text{with } d,\ c \text{ colours}$$

extends properly the colouration of X to G. Let $v \in V(G)\backslash X$. At the begining $\forall w \in N_G(v)\backslash X \quad \forall c \in \{r, g, b\}, \quad C_{v,1} = C_{v,w,c1} = (r_v \vee g_v \vee b_v)$ and $C_{v,2} = C_{v,w,c2} = (\neg r_v \vee \neg g_v \vee \neg b_v)$. Then $\forall x \in N_G(v) \cap X$ we apply property 2. and we remove $\phi(x)_v$ and $\neg\phi(x)_v$ from every clause. Since X is a dominating set of G, it exists such x. If there are two neighbours of v with different colours in X we must remove the negative clause. If there are three neighbours of v with different colours in X, the extension is impossible. Then for the remaining $w \in N_G(v)\backslash X$ we remove $C_{v,w,c}$ empty clauses. Since it remains at most 2 available colours, there must be at most 4 clauses. This ensures a correct polynomialreduction.

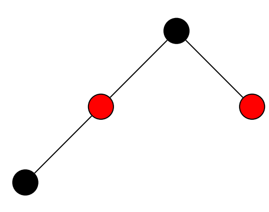
(b) *Definitions:*

- Let T be the tree created by the BFS algorithm on G.
- Let S_1 and S_2 be two vertices sets
- Let $<_a$ be the partial order on trees for ancestry

$|V(G)| \geq 2$ so there are at least two layers in T. We split the tree according to the layer number: first set S_1 is composed of even vertices layers and the second set S_2 is composed of odd vertices layers. Ensuring the following:

$$\forall v, w \in S_i \text{ s.t } layer(v) < layer(w) \text{ we have } v <_a w$$

Because T is made with the BFS algorithm, every shortest path from a node to its ancestors is in T. In other terms, edges e such that $e \in E(G)\backslash E(T)$ cannot link a node with one of its ancestors. So links between layers of the same set are all in T. So the two dominating sets are disjoint. We claim $n/2$ is a tight upper bound. It's an upper bound because we proved that one can split a graph in two dominating sets. It's tight because even filiform graphs have a minimal dominating set size of $n/2$:



(c) Let's derive an algorithm that solves 3-COLOUR:

Algorithm 2 $O^*((\sqrt{3})^n)$ algorithm to solve 3-COL problem

$T \leftarrow BFS(G)$
$S1, S2 \leftarrow bipartiteDominatingSet(T)$
if size($S1$) $<$ size($S2$) **then**
  $X \leftarrow S1$
else
  $X \leftarrow S2$
end if
$\phi \leftarrow colourize(X)$
$\phi \leftarrow reduction2SAT(X, G, \phi)$

The BFS algorithm is polynomial, defining the two dominating sets is also polynomial (question above). Then we proved that the upper bound for the smallest dominating set is $n/2$ so if we try every colouring possibility on the smallest dominating set we have $3^{\frac{n}{2}} = (\sqrt{3})^n$ possibilities, and the reduction to 2-SAT is also polynomial such as 2-SAT. So the algorithm complexity is indeed $O^*((\sqrt{3})^n)$.

Question 1.3. *Let us now attack the general k-COLOUR problem.*

(a) Let's compute all subset of vertices $Y \subseteq V(G)$ such that $\chi(G[Y]) \leq j+1$. We iterate over all subsets Y of $V(G)$ ordered following their size. If $\chi(G[Y]) \leq j$ then $\chi(G[Y]) \leq j+1$ else we remove one element y from Y and obtain $Y' := Y \backslash \{y\}$. If $\chi(G[Y']) \leq j$ then $\chi(G[Y]) \leq j+1$. If $\chi(G[Y']) \leq j+1$ then we check if adding y with any of the $j+1$ colours breaks the colouring. If it doesn't then $\chi(G[Y]) \leq j+1$. We repeat this with every element of Y. Computing all subsets of $V(G)$ costs $\sum_{t=0}^{n} \binom{n}{t}$ oplimierations and removing each element once is linear. So we could compute all subsets in time $O^*(3^n)$.

(b) Let's derive from above an algorithm that solves 3-COLOUR:

Algorithm 3 $O(3^n)$ algorithm to solve 3-COL problem

colour the graph