An Efficient Algorithm for Listing Maximal Cliques in Arbitrary Graphs

Bailey Danyluk

April 21, 2024

Abstract

The clique problem has previously been known to be solvable in exponential time (!! CITE BEST KNOWN CASE !!). In this paper, I outline new properties of maximal cliques such that they are able to be enumerated in a worst case of $\mathcal{O}(|V|^6)$.

1 Introduction

TBD

2 New properties of a clique in a given graph

In this section, a series of sets will be described that can be constructed such that they can identify cliques with only set operations on a vertex and it's neighbours. Likewise, the maximum size of every relevant set will be proved for algorithm analysis in a later section.

2.1 Preliminary definitions

Definition 1. Let G be some graph with vertices V. The set $\{C_1, C_2, \ldots, C_k\}$ where $C_i \in V$ is a member of an arbitrary maximal k-clique in G. We call this set C.

Definition 2. Let $u \in C$. I_u is the set of neighbours of u which are not in C. That is, $I_u = \{I_{u1}, I_{u2}, \ldots, I_{ui}\}$ where I_{uj} is a neighbour of u such that $I_u \cap C = \emptyset$.

Remark. The assumption that C is maximal can be made since C is not constructed; it is just a subset that must exist within the neighbours of some $u \in V$. If C is not maximal, we can remove the missing vertices from I_u and add them to C.

Lemma 2.1. *Let $u, v \in C$ where $(I_u \cup I_v) \neq \emptyset$. There must exist $w \in C$ such that $I_w \cap (I_u \cap I_v) = \emptyset$.*

Proof. If we were unable to pick $w \in C$ such that $I_w \cap (I_u \cap I_v) = \emptyset$, that would mean that there exists a vertex in V which every vertex in C has an edge to that is not in C. This means that C is not maximal, or $w \in C$. Therefore, there must be at least one $w \in C$ such that I_w is disjoint from $I_u \cap I_v$. $\square$

Definition 3. Let $u \in C$. N_u are all vertices reachable in at most a single step from u. That is, $N_u = I_u \cup C$.

Note. This is different than just all vertices adjacent to u. $u \in N_u$ since $u \in C$. This follows from the definition since u is zero steps from itself.

Corollary 2.2. *Let* $u \in V$. $|N_u| = deg(u) + 1$.

Corollary 2.3. $\forall u \in V$, $|N_u| \leq |V|$.

Remark. Every vertex in a graph is apart of some clique. This clique may just be K_1 or K_2, but these are maximal cliques which need to be counted to properly create the list.

2.2 Common Neighbours of a Vertex

Definition 4. Let $u \in V$. CN_{ui} is the set of vertices in N_u which u shares with it's i-th neighbour N_{ui}.

Definition 5. CN_u is the set which lists all neighbours which u has in common with any of it's neighbours. That is, if u can reach a vertex v and some neighbour of u can also reach the vertex v, then v exists within some subset of CN_u.

We can construct CN_u by intersecting N_u with the neighbours of u's i-th neighbour.

$$CN_u = \bigcup_{i=1}^{\deg(u)} \{N_u \cap N_v\}, v = N_{ui} \tag{1}$$

Lemma 2.4. *Let* $u, v \in C$ *where* $u \neq v$ *and* $I_u \cap I_v = \emptyset$. *Then* $CN_{ui} \cap CN_{vj} = C$ *for any* i, j.

Proof. Let $u, v \in C$ such that $u \neq v$. $CN_u \cap CN_v = (I_u \cup C) \cap (I_v \cup C)$. Since $I_u \cap I_v = \emptyset$, we can see that the previous statement is equal to $C = C \cup (I_u \cap I_v) = C \cup \emptyset$. $\square$

Corollary 2.5. *For some* $u \in V$, CN_u *only contains vertices which* u *is adjacent to.*

Corollary 2.6. $\forall u \in V$, $|CN_u| = deg(u)$

Corollary 2.7. $\forall u \in V$, $|CN_u| \leq |V| - 1$

So, CN_u is a set of neighbours which u has in common with an adjacent vertex v. This set will be used in the next section to create another set which will list all edges which u shares with all of it's neighbours.

2.3 Common Edges of a Vertex

Definition 6. Let $u \in V$. CE_{ui} is the set of edges in N_u which u shares with it's i-th neighbour N_{ui}.

We can construct CE_{ui} by intersecting CN_{ui} with the j-th neighbour's common neighbours, $\forall CN_{(N_{uj})}$. When we take the union of all of these intersections, we will have constructed CE_u.

$$e(u,n,i,j) = \begin{cases} CN_{ui} \cap CN_{nj}, n \in N_u, & \text{if } u \in (CN_u \cap CN_n) \\ & \quad \text{and } n \in (CN_u \cap CN_n) \\ \emptyset, & \text{otherwise} \end{cases} \tag{2}$$

Remark. We do have to check if both vertices are in the resulting intersection. Consider a path graph, and select a vertex, u, not on the ends. Both neighbours of this path graph will attempt to intersect their neighbours $\neq u$ with u. The resulting set will be the set that only contains the neighbour. This doesn't fit our definition of CEu_i, so we ignore these values.

Lemma 2.8. *Equation (2) will be a set that strictly contains vertices that both $u \in V$ and it's k-th neighbour have an edge to.*

Proof. Let $u \in V$, $E_u = CN_{ui}$ and $E_v = CN_{vj}$ where $v = N_{uk}$ for some i, j. Because u and v are neighbours, an edge exists between u and all vertices in E_u from (2.5). Likewise, the same argument applies for v and E_v. When we intersect $E_u \cap E_v$, the resulting set will be vertices that both u and v have an edge to. □

Definition 7. Let $u \in V$. CE_u is the set which contains sets of all edges, $e(u,n,i,j)$, which u has in common with all of it's neighbours $n \in N_u$.

Lemma 2.9. *For some $u \in V$*

$$CE_u = \bigcup_{i=1}^{|CN_u|} \bigcup_{k=1}^{|N_u|} \{n = N_{uk} \mid \bigcup_{j=1}^{|CN_n|} \{e(u,n,i,j)\}\} \tag{3}$$

Proof. Let $u \in V$. What this abuse of notation does is: for the i-th iteration, we take the i-th common neighbour of u and test it against every neighbours common neighbours using the function we defined in (2). So, we will get a set of common edges that both u and every neighbour in N_u have. This is the definition of CE_u. □

Lemma 2.10. *Let $u \in V$. $|CE_u| \leq |V|^3$*

Proof. We know that for any $u \in V$, $|CN_u| = |V|$ from (2.7). Likewise, the maximum size of $N_u = |V|$ as well from (2.3). So, (2.9) will take at most $|V| \cdot |V| \cdot |V| = |V|^3$ intersections to complete, only ever constructing $|V|^3$ sets that all get unioned into the main CE_u set.

$\therefore |CE_u| \leq |V|^3$. $\square$

Theorem 2.11. $u \in C \iff C \in CE_u$.

Proof of $u \in C \implies C \in CE_u$. Let $u, v \in C$ where $u \neq v$.

Note. $C \in N_u$ and $C \in N_v$ since u and v are chosen from the same clique.

There are two cases to consider:

Case (I_u has no elements in common with I_v). So, $I_u \cap I_v = \emptyset$.

From (2.4) we know that $C \in CN_{ui}$ and $C \in CN_{vj}$. When we construct CE_u, we will intersect $CN_{ui} \cap CN_{vj} = C$.

$\therefore C \in CE_u$.

Case (I_u has at least one element in common with I_v). Pick some $w \in C$ that has no non-clique neighbours in common with u and v. This is possible because of (2.1). Let $CN_{ui} = (I_u \cup C) \cap (I_v \cap C)$ (From construction of CN, (1)). This means that $\exists j \leq \deg(w) \mid CN_{ui} \cap CN_{wj} = C$.

$\therefore$ By construction of CE_u, $C \in CE_u$.

$\square$

Proof of $u \in C \impliedby C \in CE_u$. Let $u \in V$. Assume $C \in CE_u$, but $u \notin C$. Pick $v \in C$.

Case ($v \in N_u$). We will pick another $v \in C$ such that $v \notin N_u$. If we cannot pick such a v, that means that C is not maximal, or $u \in C$.

Case ($v \notin N_u$). From (2.5), $\forall i \leq CN_u, v \notin CN_{ui}$. However, for $C \in CE_u$, we must intersect CN_u with one of it's neighbours CN_{N_u} so that C is created. But since v is not in CN_u at all, we would be missing v, so $(C \setminus v) \in CE_u$. This is a contraction that $C \in CE_u$.

$\therefore u$ must have an edge to v. However, if we create this edge, then $u \in C$.

From the two cases, we see if $C \in CE_u$ then $u \in C$. $\square$

CE_u is the set which lists all edges which u has in common with all of it's neighbours. That is, if u has an edge to vertex v and some neighbour of u also has an edge to the vertex v, then v exists within some subset of CE_u. We know that CE_u's length is bounded by a polynomial (2.10), and that only vertices in a clique C will contain $C \in CE_u$. Using these facts, in the next section we will develop an algorithm to list all cliques in a graph.

3 An algorithm to list maximal cliques in an arbitrary graph

In this section I will present an algorithm in multiple parts which, when combined, will list all maximal cliques in an arbitrary graph.

3.1 Algorithm

Algorithm 3.1: Create Common Neighbour Set

```
function: list_common_neighbours
    input: Set of vertices V, Set of tuples (v_i, v_j) called E
    output: Set of common neighbours
    begin
        edge_reports ← {}
        for v in V
            neighbours ← adjacent vertices of v
            for n in neighbours
                edge ← the edge (v, n) ∈ E
                if edge in edge_reports
                    edge_reports[edge][2] ← neighbours
                else
                    edge_reports[edge] ← {neighbours, ∅}
                end
            end
        end

        common_neighbours = {}
        for edge in E
            ce ← edge_reports[edge][1] ∩ edge_reports[edge][2]
            add ce to set common_neighbours[edge[1]]
            add ce to set common_neighbours[edge[2]]
        end

        return common_neighbours
    end
```

Algorithm 3.2: Create Common Edge Set

```
function: list_common_edges
    input: Set of vertices V, Set of tuples (v_i, v_j) called E
    output: Set of common edges
    begin
        common_neighbours ← common_neighbours(V, E)
        common_edges ← {}

        for u in V
            for v in neighbour vertices of u
                for Ucn in common_neighbours[u]
                    for Vcn in common_neighbours[v]
                        let ce = Ucn ∩ Vcn
                        if u in ce and v in ce
                            add ce to set common_edges[u]
                        end
                    end
                end
            end
        end

        return common_edges
    end
```

Algorithm 3.3: List Maximal Cliques

```
function: list_maximal_cliques
    input: Set of vertices V, Set of tuples (v_i, v_j) called E
    output: Set of maximal cliques in the graph (V, E)
    begin
        common_edges ← common_edges(V, E)

        cliques ← {}
        for u in V
            for Uce in common_edges[u]
                clique ← {u}
                for n in neighbours of V
                    if Uce ∈ common_edges[n]
                        add n to clique
                        remove Uce from common_edges[n]
                    end
                end

                add clique to cliques
                remove Uce from common_edges[n]
            end
        end

        // Remove sub-cliques so we are left with only maximal
        for c1 in cliques
            for c2 in cliques
                if c1 ⊂ c2
                    remove c1 from cliques
                end
            end
        end

        return cliques
    end
```

Remark. The reason non-maximal cliques exist is that the properties listed in the previous section show that the maximal clique must exist, but are not necessarily exclusive, in CE. Using the algorithm as described, it will never misreport a clique since

4 Conclusion

Corollary 4.1. $P = NP$

Proof. Proof is left as an exercise to the reader. □