Maximum Clique Project

DynamicPookiemon

December 1, 2025

Contents

1 Introduction **3**
1.1 Complexity Overview . . . 3

2 Problem Statement **3**
2.1 Decision Version . . . 3
2.2 Optimization Version . . . 3
2.3 Maximum vs. Maximal Clique . . . 4

3 Proof that Decision Problem for Clique is NP-Complete **4**
3.1 CLIQUE is in NP . . . 4
3.2 CLIQUE is NP-Hard . . . 4
3.2.1 The Reduction Algorithm (Construction) . . . 5
3.2.2 Proof of Equivalence . . . 5
3.2.3 Complexity Analysis . . . 6

4 Reductions and Equivalence **6**
4.1 Solving Independent Set using a Solver for Clique . . . 6
4.2 Why the Decision and Optimization Problems are Equivalent . . . 7
4.2.1 Solving the optimization problem using a solution for decision problem . 7
4.2.2 Solving the decision problem using a solution for optimization problem . 7

5 Brute Force Algorithm (for Decision Problem) **8**
5.1 Intuition . . . 8
5.2 Proof of Correctness . . . 8
5.3 Time Complexity . . . 9

6 Optimized Exact Algorithm: Greedy + Core Pruning + Tomita **10**
6.1 Intuition . . . 10
6.2 Definitions . . . 11
6.3 Claim . . . 11
6.4 The Algorithm . . . 11
6.5 Complexity Analysis . . . 11

7 Proof 1: Correctness of the ILP Formulation **20**
7.1 Problem Definitions . . . 20

8 Proof 2: The LP Relaxation is an Upper Bound **21**
8.1 Definitions . . . 21

9 Introduction **22**

10 Given: Random Graph **22**

11 Goal: Maximum Clique **22**

12 Exploring the graph for goal: Tomita's Algorithm **22**

13 The Barrier: The Overlap Gap Property (OGP) **23**

14 The Failure: Why OGP Breaks the Pivot **23**

15 Brief summary of the rigorous result in PNAS 2021 **24**

15.1 Step 1: The Counter 24

15.2 Step 2: The Expectation (The U-Curve) 24

15.3 Step 3: The Conclusion 24

16 Conclusion **25**

1 Introduction

One of the main and most studied problems in graph theory and theoretical computer science is the **maximum vertex clique problem**. The problem was one of the 21 computational problems identified and proven NP-complete in 1972 by Richard Karp famously. It is central to the study of combinatorial optimization and it is closely related to other areas like network analysis, bioinformatics, social network theory, and computational complexity.

The problem in its simplest form reflects the concept of identifying a subset of entities (modeled as the nodes of a graph) that are fully interconnected. This helps to identify a close-knit community or cluster in which all members are directly related to all other members. It is also the dual problem to the graph coloring problem.

1.1 Complexity Overview

The vertex clique problem is an **NP-complete** problem. This means:

- There is no known polynomial-time algorithm to solve it for all general graphs.
- If someone finds a fast (polynomial-time) solution to the clique problem, it would imply a solution to a wide range of other hard problems (proving $P = NP$).

2 Problem Statement

Given an undirected graph $G = (V, E)$, where:

- V is the set of vertices (nodes), and
- E is the set of edges (formally, $E \subseteq \{(u, v) \mid u, v \in V, u \neq v\}$).

A **clique** is defined as a subset of vertices $C \subseteq V$ such that every pair of distinct vertices in C is connected by an edge in E. In other words, the subgraph induced by C is a complete graph.

More formally, a clique is a set of vertices $C \subseteq V$ such that for every pair of distinct vertices $u, v \in C$, the edge $(u, v) \in E$. The size of the clique is defined to be $|C|$ (cardinality of C).

There are two primary versions of the problem:

2.1 Decision Version

Given a graph G and an integer k, does there exist a clique of size at least k?

- **Output:** YES or NO.

2.2 Optimization Version

Find the largest clique in the graph (called the maximum clique). Formally, the maximum clique is $C_{\max} \subseteq V$ such that:

$$|C_{\max}| = \max\{|C| \mid C \text{ is a clique in } G\}$$

The value $|C_{\max}|$ is called the **clique number** of the graph, denoted $\omega(G)$.

- **Output:** The size of the largest clique or the clique itself.

2.3 Maximum vs. Maximal Clique

A crucial distinction to know is the difference between maximum clique and maximal clique. A maximum clique is definitely a maximal clique, but the converse is (in general) not true.

Maximum Clique: A maximum clique is the largest possible clique in the entire graph (globally optimal). It is the largest clique that exists in the graph—no clique has more vertices than this one.

$$|C| = \omega(G) = \max\{|C'| \mid C' \text{ is a clique in } G\}$$

Maximal Clique: A maximal clique is a clique that cannot be extended by adding any other adjacent vertex (locally optimal). You can't add another vertex to it without losing the clique property, but it may not be the largest possible clique in the graph. A clique C is maximal if there is no vertex $v \in V$ and $v \notin C$ such that adding v to the clique still keeps it a clique.

3 Proof that Decision Problem for Clique is NP-Complete

To prove that the CLIQUE problem is NP-complete, we must satisfy two conditions:

1. **CLIQUE is in NP:** We can verify a potential solution in polynomial time.
2. **CLIQUE is NP-hard:** Every problem in NP can be reduced to CLIQUE in polynomial time. We prove this by reducing a known NP-complete problem to CLIQUE.

3.1 CLIQUE is in NP

To show CLIQUE is in NP, we must show that a given "certificate" (a potential solution) can be verified in polynomial time.

Certificate: A subset of vertices $C' \subseteq V$ that we claim is a clique of size k.

Verifier: We can build an algorithm to verify this certificate:

1. Check if the size of C' is equal to k. This takes $O(k)$ time, which is at most $O(|V|)$.
2. Iterate through every pair of distinct vertices (u, v) in C'. The number of pairs is $\binom{k}{2} = \frac{k(k-1)}{2}$, which is $O(k^2)$ or at most $O(|V|^2)$.
3. For each pair, check if the edge (u, v) exists in the graph's edge set E. An edge lookup (using an adjacency matrix, for example) takes $O(1)$ (or $O(|V|)$ depending on the structure), but is always polynomial.
4. If all pairs have an edge between them, the verifier accepts C' as a valid solution. If any pair is missing an edge, it rejects C'.

Since this entire process runs in polynomial time relative to the size of the input graph G, CLIQUE is in NP.

3.2 CLIQUE is NP-Hard

We will show that 3-SAT $\leq_p$ CLIQUE. This means we will take an arbitrary instance of a 3-SAT formula and construct a graph G and an integer k such that G has a clique of size k if and only if the 3-SAT formula is satisfiable.

3.2.1 The Reduction Algorithm (Construction)

Let ϕ be a 3-SAT formula with m clauses and n variables.

$$\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$$

Each clause C_r contains exactly 3 literals (e.g., $x_1 \vee \neg x_2 \vee x_3$).

1. **Construct Vertices (V):** For every literal in every clause, create a vertex in the graph. Since there are m clauses with 3 literals each, the graph will have $3m$ vertices. We can label a vertex as (l, r), representing the literal l inside clause C_r.
2. **Construct Edges (E):** We draw an edge between two vertices (l, r) and (l', s) if and only if:
   - They are in different clauses: $r \neq s$. (We never connect literals within the same clause).
   - They are consistent: $l \neq \neg l'$. (We never connect a literal to its negation, e.g., we do not connect x_1 to $\neg x_1$).
3. **Define k:** Set the target clique size $k = m$ (the number of clauses).

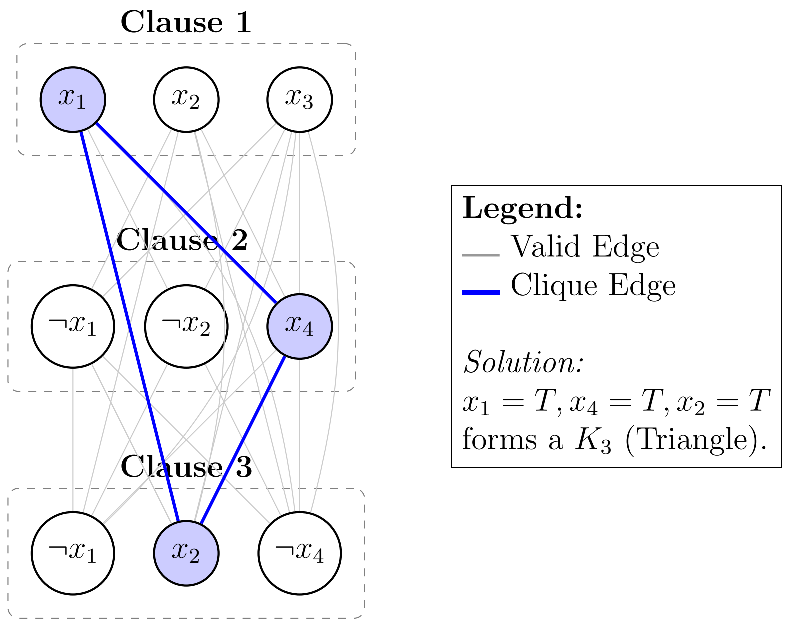

Figure 1: Visualization of 3-SAT $((x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee \neg x_4))$ to CLIQUE Reduction. The blue triangle identifies a set of consistent literals.

3.2.2 Proof of Equivalence

We must prove the logical equivalence in both directions:

$$\phi \text{ is satisfiable} \iff G \text{ has a clique of size } m$$

Direction 1: If ϕ is satisfiable $\rightarrow$ G has a clique of size m.

- If ϕ is satisfiable, there exists a truth assignment that makes every clause $C_1 \ldots C_m$ true.
- This means in every clause C_r, at least one literal evaluates to True.

- Select exactly one true literal from each of the m clauses. This gives us a set of m vertices.
- Are they connected?
  - They are all from different clauses (by selection).
  - They are consistent (since a valid truth assignment cannot make both x and $\neg x$ true simultaneously).
- Therefore, all m vertices are connected to each other, forming a clique of size m.

Direction 2: If G has a clique of size $m \to \phi$ is satisfiable.

- Suppose there exists a clique K of size m.
- Because vertices within the same clause are not connected, the clique cannot contain more than one vertex from any single clause.
- Since the clique size is exactly m and there are m clauses, the clique must contain exactly one vertex per clause.
- Because these vertices form a clique, there are edges between all of them. By our construction, edges only exist between consistent literals (no x connected to $\neg x$).
- We can set the variables corresponding to these m literals to True. Since there are no contradictions, this is a valid assignment.
- This assignment satisfies every clause, meaning ϕ is satisfiable.

3.2.3 Complexity Analysis

To verify this is a valid reduction, the transformation must happen in polynomial time.

- **Vertices:** We create $3m$ vertices. This is linear in the size of the input.
- **Edges:** We check every pair of vertices. There are $O(m^2)$ pairs. Comparing literals takes constant time.
- **Total Time:** The construction takes $O(m^2)$ time, which is polynomial.

Conclusion: Since CLIQUE is in NP (Part 1) and 3-SAT reduces to CLIQUE (Part 2), we conclude that CLIQUE is NP-complete.

4 Reductions and Equivalence

4.1 Solving Independent Set using a Solver for Clique

A common and direct reduction from the **Independent Set (IS)** problem to Clique. We need to transform any instance of Independent Set (G, k) into an instance of CLIQUE (G', k') in polynomial time, such that the answer to both instances is the same.

Given the IS instance (G, k), we construct a new CLIQUE instance $(\overline{G}, k)$.

- The graph $\overline{G}$ is the complement graph of G.
- $\overline{G}$ has the same set of vertices V as G.
- An edge (u, v) exists in $\overline{G}$ if and only if the edge (u, v) does not exist in G.
- The integer k remains the same.

This transformation is polynomial. If G is represented by an adjacency matrix, we can create the adjacency matrix for $\overline{G}$ by flipping all 0s to 1s and 1s to 0s (ignoring the diagonal). This takes $O(|V|^2)$ time.

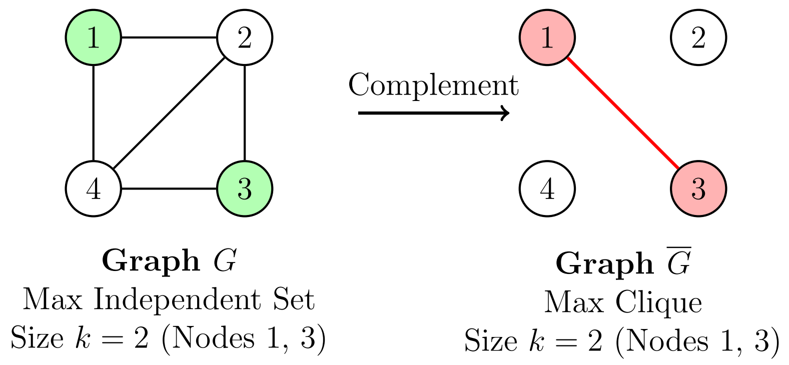

Figure 2: Reduction from Independent Set to Clique. The non-adjacent vertices in G (green) become fully connected vertices in $\overline{G}$ (red).

Proof of Equivalence: We must show that G has an independent set of size k if and only if $\overline{G}$ has a clique of size k.

- ($\Longrightarrow$) **Forward:** Assume G has an independent set I of size k. Let u and v be any two distinct vertices in I. By the definition of an independent set, the edge (u, v) does not exist in E. By our reduction, if $(u, v) \notin E$, then the edge (u, v) must exist in $\overline{G}$. Since this is true for all pairs of vertices in I, I is a clique of size k in $\overline{G}$.
- ($\Longleftarrow$) **Backward:** Assume $\overline{G}$ has a clique C of size k. Let u and v be any two distinct vertices in C. By the definition of a clique, the edge (u, v) must exist in $\overline{G}$. By our reduction, if (u, v) is in $\overline{G}$, then the edge (u, v) does not exist in E. Since this is true for all pairs of vertices in C, no two vertices in C are connected in G. Therefore, C is an independent set of size k in G.

4.2 Why the Decision and Optimization Problems are Equivalent

4.2.1 Solving the optimization problem using a solution for decision problem

Given $G(V, E)$, let $|V| = n$.

```
for $k = n$ down to 1 do
    if there is a clique of size $k$ in $G$ then
        return $k$
    end if
end for
```

As the above algorithm iterates from n (theoretical maximum size of clique for G) down to 1 (minimum size of clique for G), it returns the size of the clique with maximum size.

4.2.2 Solving the decision problem using a solution for optimization problem

Given $G(V, E)$ and k, let $|V| = n$.

```
Let $\omega$ = size of maximum clique in $G$
if $0 \le k \le \omega$ then
    return true
else
```

```
    return false
end if
```

Why this works: Let ω be the clique number. Then, we have a clique C of size ω. We show we also have a clique of size k where $0 \leq k \leq \omega$. Remove any $\omega - k$ vertices from C to get C'; this is a clique of size k in G. Moreover, by definition, C is the maximum clique of G, so there exists no clique in G of size more than ω. Hence the reduction is correct.

5 Brute Force Algorithm (for Decision Problem)

5.1 Intuition

If possible, let there be a clique of size k. Pick an arbitrary vertex u in V. It either is in the clique or not. Recursively solve for both:

1. If u belongs to the clique, delete all vertices not connected to u, then run the algorithm on the remaining graph looking for a clique of size $k - 1$.
2. If u is not in the clique, then delete u along with all edges connected to u, looking for a clique of size k.

Return the "OR" of both of these results.

Algorithm 1 HasClique(G, k)

```
Require: Graph $G = (V, E)$ and integer $k$
Ensure: True if $G$ contains a clique of size $k$, False otherwise
 1: function HasClique($G, k$)
 2:     if $k == 0$ then                    ▷ Base Case 1: A clique of size 0 always exists
 3:         return True
 4:     end if
 5:     if $|V(G)| < k$ then                       ▷ Base Case 2: Graph too small
 6:         return False
 7:     end if
 8:     Let $u$ be an arbitrary vertex in $G$
                                                ▷ Case 1: Assume u IS in the k-clique
 9:     Let $N(u)$ be the set of neighbors of $u$
10:     Let $G_1$ be the subgraph induced by $N(u)$
11:     if HasClique($G_1, k - 1$) == True then
12:         return True
13:     end if
                                            ▷ Case 2: Assume u IS NOT in the k-clique
14:     Let $G_2 = G - \{u\}$
15:     if HasClique($G_2, k$) == True then
16:         return True
17:     end if
18:     return False
19: end function
```

5.2 Proof of Correctness

We prove the correctness of the HasClique algorithm using strong induction on the number of vertices $n = |V|$.

Proof. Let $P(n)$ be the proposition that the algorithm correctly determines the existence of a clique of size k for any graph with n vertices.

Base Cases:

- If $k = 0$, the algorithm returns **True**. This is correct as the empty set is formally a clique of size 0 (or typically any single node is a clique of size 1, but algorithmically $k = 0$ is the stopping condition for success).
- If $n < k$, the algorithm returns **False**. This is correct because a graph with n vertices cannot contain a subset of k distinct vertices if $n < k$.

Inductive Step: Assume $P(m)$ is true for all graphs with $m < n$ vertices (Inductive Hypothesis). We must show that $P(n)$ holds.

Consider an arbitrary graph G with n vertices and a target size k. The algorithm picks an arbitrary vertex u and explores two possibilities. We must prove that G has a clique of size k if and only if the algorithm returns **True**.

Direction 1 ($\Longleftarrow$): If the algorithm returns True, a clique exists. Since the algorithm uses an OR condition, it returns True if either recursive call returns True.

1. **Case 1:** $\textsc{HasClique}(G_1, k-1)$ returns True. Here, G_1 is the subgraph induced by neighbors $N(u)$. Since $|V(G_1)| < n$, by the inductive hypothesis, there exists a clique $C' \subseteq N(u)$ of size $k-1$. Because every vertex in C' is a neighbor of u, adding u to C' creates a set $C = C' \cup \{u\}$ where u is connected to all nodes in C'. Thus, C is a clique of size k in G.
2. **Case 2:** $\textsc{HasClique}(G_2, k)$ returns True. Here, $G_2 = G - \{u\}$. Since $|V(G_2)| = n-1 < n$, by the inductive hypothesis, there exists a clique $C \subseteq G_2$ of size k. Since G_2 is a subgraph of G, C is also a clique of size k in G.

Direction 2 ($\Longrightarrow$): If a clique exists, the algorithm returns True. Assume there exists a clique K in G of size k. Consider the arbitrary vertex u chosen by the algorithm. There are only two possibilities:

1. **Scenario A ($u \in K$):** If u is part of the clique K, then the remaining vertices $K' = K \setminus \{u\}$ form a clique of size $k-1$. Since K is a clique, all vertices in K' must be neighbors of u. Therefore, K' exists entirely within the subgraph induced by $N(u)$ (which is G_1). By the inductive hypothesis, the first recursive call will find this and return **True**.
2. **Scenario B ($u \notin K$):** If u is not part of the clique K, then all vertices of K are in the set $V \setminus \{u\}$. Therefore, the clique K exists entirely within $G_2 = G - \{u\}$. By the inductive hypothesis, the second recursive call will find this and return **True**.

Since the algorithm returns the logical OR of these two cases, it is guaranteed to return **True** if a clique exists. Thus, the algorithm is correct. $\square$

5.3 Time Complexity

Let's analyze the running time $T(n, k)$, where $n = |V|$ is the number of vertices.

Recurrence Relation: The algorithm makes two recursive calls at each step.

- **Case 1:** HasClique$(G_1, k-1)$. The subgraph G_1 is induced by the neighbors of u. In the worst case, u could be connected to all other $n-1$ vertices. So, this call is on a graph of up to $n-1$ vertices, searching for a clique of size $k-1$. This gives us $T(n-1, k-1)$.

- **Case 2:** HasClique(G_2, k). We remove one vertex, u. This call is on a graph of $n-1$ vertices, searching for a clique of size k. This gives us $T(n-1, k)$.

The work done at each step (selecting a vertex, creating subgraphs) is polynomial in n, let's say $O(n^2)$. This gives the recurrence relation:

$$T(n, k) = T(n-1, k-1) + T(n-1, k) + O(n^2)$$

Solving the Recurrence: This recurrence is very similar to the one for binomial coefficients, which is $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$. The number of leaf nodes in the recursion tree will be $\binom{n}{k}$.

Final Complexity: The total time is the number of recursive calls multiplied by the work done in each call.

- Number of calls: $O(\binom{n}{k})$
- Work per call: $O(n^2)$ (for creating the subgraphs)

Therefore, the total time complexity is:

$$O\left(\binom{n}{k} \cdot n^2\right)$$

Since $\binom{n}{k}$ is $O(n^k)$, the complexity can also be expressed as $O(n^{k+2})$. This is polynomial for a fixed k, but exponential if k is part of the input.

6 Optimized Exact Algorithm: Greedy + Core Pruning + Tomita

While the Maximum Clique problem is NP-hard, we can significantly reduce the search space for practical instances by combining heuristic lower bounds with graph reduction techniques. This section introduces a hybrid algorithm that uses a **Greedy heuristic** to find a lower bound, **k-core decomposition** to prune the graph, and the **Tomita algorithm** (an optimized backtracking approach) to find the exact solution.

6.1 Intuition

The search space for a maximum clique algorithm grows exponentially with the number of vertices. However, vertices that are sparsely connected are unlikely to be part of a large clique. Such vertices can be safely removed.

1. **Lower Bound (LB):** First, we quickly find a large (but not necessarily maximum) clique using a fast greedy algorithm. The size of this clique gives us a lower bound LB. We know the maximum clique size $\omega(G) \geq LB$.
2. **Condition:** Any vertex that belongs to a clique of size k must have a degree of at least $k-1$ within that clique. If we are looking for a maximum clique (which must be size $\geq LB$), we can safely remove any vertex that cannot possibly be part of a clique of size LB.
3. **Exact Search:** We run the heavy-duty exact solver (Tomita) only on the much smaller, denser remaining graph.

6.2 Definitions

To further develop on this intuition, we define the following:
Definition 1 (k-Core): A k-core of a graph G is a maximal subgraph $H \subseteq G$ in which every vertex has a degree at least k within H.
Definition 2 (Core Number): The core number of a vertex v, denoted $core(v)$, is the largest integer k such that v belongs to a k-core.

6.3 Claim

We claim that pruning (removing) vertices with $core(v) < LB - 1$ does not discard any vertex in the maximum clique if $LB \leq \omega(G)$.

Proof. Let C^* be a maximum clique in G, with size $\omega(G) = |C^*|$.

Consider any vertex $v \in C^*$. Since C^* is a clique of size $\omega(G)$, v is connected to every other vertex in C^*. Therefore, the degree of v in the subgraph induced by C^* is exactly $\omega(G) - 1$.

By the definition of a k-core, since every vertex in C^* has degree $\omega(G) - 1$ within C^*, the set C^* forms a subgraph where minimum degree is $\omega(G) - 1$. Thus, C^* is a subset of the $(\omega(G) - 1)$-core.

Consequently, for every $v \in C^*$, the core number $core(v) \geq \omega(G) - 1$. Since $\omega(G) \geq LB$, it follows that:

$$core(v) \geq LB - 1$$

Therefore, removing nodes where $core(v) < LB - 1$, removes no vertices belonging to the maximum clique C^*. The maximum clique is fully contained in the reduced graph G'. □

Specifically, if we find any valid lower bound, LB, on the size of the clique (say, by using a greedy algorithm: assuming the vertex with maximum degree is in the clique and finding the size of this clique), we can prune vertices with a core number less than $LB - 1$ and run any algorithm on the reduced graph G' to find the maximum clique.

Then, the claim shows that $\omega(G') \geq \omega(G)$ (since the maximum clique is preserved). However, since G' is a subgraph of G, we also know that $\omega(G') \leq \omega(G)$. It follows that $\omega(G') = \omega(G)$. Hence, running the algorithm on G' finds the maximum clique of G.

Finally note that, the larger this lower bound, the smaller the graph becomes, and the faster the algorithm can run.

6.4 The Algorithm

6.5 Complexity Analysis

The complexity is split into three stages:

1. **Greedy Heuristic:** Sorting vertices takes $O(|V| \log |V|)$. Iterating and checking adjacency takes $O(|V|^2)$ or $O(|E|)$.
2. **k-Core Decomposition:** The core decomposition can be computed in linear time $O(|V| + |E|)$ using the peeling algorithm (iteratively removing the lowest degree vertex).
3. **Tomita Algorithm:** The worst-case complexity of the Tomita algorithm (Bron-Kerbosch with pivoting) is $O(3^{|V'|/3})$, where $|V'|$ is the number of vertices in the *reduced* graph.

Algorithm 2 Optimized MaxClique (Greedy + Core Pruning + Tomita)

```
Require: Graph $G = (V, E)$
Ensure: The Maximum Clique $C_{max}$
1: Step 1: Compute Lower Bound (Greedy)
2: $C_{greedy} \leftarrow \emptyset$
3: Sort vertices $V$ by degree (descending)
4: for $v \in V$ do
5:     if $v$ is connected to all $u \in C_{greedy}$ then
6:         $C_{greedy} \leftarrow C_{greedy} \cup \{v\}$
7:     end if
8: end for
9: $LB \leftarrow |C_{greedy}|$
10: Step 2: Pruning using $k$-core
11: Compute core numbers for all $v \in V$
12: $V' \leftarrow \{v \in V \mid core(v) \geq LB - 1\}$
13: $G' \leftarrow$ Subgraph induced by $V'$
14: Step 3: Exact Search (Tomita)
15: $C_{exact} \leftarrow$ TOMITA($G'$)        ▷ Run standard Tomita/Bron-Kerbosch on reduced graph
16: Step 4: Return Maximum
17: if $|C_{exact}| > |C_{greedy}|$ then
18:     return $C_{exact}$
19: else
20:     return $C_{greedy}$
21: end if
```

Overall Complexity: $O(3^{|V'|/3})$. While the asymptotic worst-case remains exponential, the practical speedup is significant because $|V'|$ (the size of the dense core) is often much smaller than $|V|$ (the total graph size). For real-world scale-free networks (like social networks), the core is small, making this approach extremely effective.

I. Basic Bron-Kerbosch Algorithm

Definitions

Definition 1 (Clique)**.** A **Clique** of $G(V, E)$ is $C \subseteq V$ s.t. the subgraph of Graph G induced by C is complete. That is, $\forall v, \omega \in C, (v, \omega) \in E$.
Definition 2 (Maximal Clique)**.** A Clique C of $G(V, E)$ is **maximal** if: there is no Clique D of G s.t. $C \subseteq D$ but $C \neq D$.
Definition 3 (Maximum Clique)**.** The Clique C^* is **maximum** s.t. $|C^*| \geq |C|, \forall C$ s.t. C is a clique of G.
Notation: $\omega(G) \to |C^*|$

Algorithm

For a graph $G(V, E)$:

```
1: procedure BK(R, P, X)
2:     if P and X are empty then
3:         report R as a maximal clique
4:     end if
5:     for $v \in P$ do
6:         BK($R \cup \{v\}$, $P \cap n(v)$, $X \cap n(v)$)
7:         $P \leftarrow P \setminus \{v\}$
8:         $X \leftarrow X \cup \{v\}$
9:     end for
10: end procedure
```

Correctness

Claim 1. If we call BK($\{\}, V, \{\}$), the algorithm will report all maximal cliques exactly once.

Proof. The proof has four parts:

1. **Show all R reported are cliques.**
   Let $R \subseteq V$ be what the algorithm reports. For contradiction, assume R is not a clique: $\Rightarrow \exists v, \omega \in R$ s.t. $(v, \omega) \notin E \Rightarrow \omega \notin n(v)$.

   Let v be the vertex that was added to R first (among all non-adjacent pairs in R) at recursion step i, where R was R_i and P was P_i. In the next recursion call: $P_{i+1} = P_i \cap n(v)$.

   Since we assumed v was added first, ω must have been in P_i. Since $\omega \notin n(v)$, $\omega \notin P_i \cap n(v) \Rightarrow \omega \notin P_{i+1}$. Since we never add vertices back to P, ω can never be added to R (in this branch). $\Rightarrow \omega \notin R$, which contradicts our assumption. $\therefore$ R is a Clique.

2. **Show all R reported are maximal.**
   Now for contradiction, assume R isn't a maximal Clique. $\Rightarrow \exists v \in V \setminus R$ such that $(v, \omega) \in E, \forall \omega \in R$. This means $\forall \omega \in R, v \in n(\omega)$ — ①

   Let $R = \{v_1, v_2, \ldots, v_p\}$. When R is reported, the sets P and X must be empty. The final P set for this branch is $P_f = P_0 \cap n(v_1) \cap n(v_2) \cap \cdots \cap n(v_p)$, where $P_0 = V$. Clearly $v \in P_0$ [since $P_0 = V$].

By ①, $v \in n(v_1), v \in n(v_2), \ldots, v \in n(v_p)$. $\Rightarrow v \in P_0 \cap n(v_1) \cap n(v_2) \cap \cdots \cap n(v_p) \Rightarrow v \in P_f$.

This means $P_f \neq \phi$, which contradicts the condition for reporting R (that P must be empty). (A similar argument holds for $v \in X_0$, which would imply $X_f \neq \phi$). So, R must be a maximal Clique.

3. **Show all maximal cliques M are found.**
   Let M be some maximal Clique in the graph. We must show M is reported by some branch. Let the vertices of V be ordered $v_1, v_2, \ldots, v_k$ in the order they are picked by the main for loop in the initial call. Let v_i be the first vertex in this ordering such that $v_i \in M$. This means $\forall \omega \in R, v \in n(\omega)$ — (1) $R = \{\}, P = \{v_i, v_{i+1}, \ldots, v_k\}$, $X = \{v_1, v_2, \ldots, v_{i-1}\}$ (Note: $\forall v_j \in X, v_j \notin M$ by our choice of v_i)

   The algorithm calls BK($\{v_i\}$, $P \cap n(v_i)$, $X \cap n(v_i)$). We know $M \setminus \{v_i\} \subseteq P \cap n(v_i)$ because M is a clique and v_i was the first vertex of M in the ordering.

   This process will continue, recursively selecting vertices from M. Let the set of vertices chosen be $M = \{\omega_1, \ldots, \omega_m\}$. When all are chosen, $R_f = M$. We must show $P_f = \phi$ and $X_f = \phi$.

   a) Show $P_f = \phi$: Assume $P_f \neq \phi \Rightarrow \exists v \in P_f$. This means v is adjacent to every $\omega_i \in M$. This implies $M \cup \{v\}$ is a valid clique (since v connects to all of M and all of R). Since $v \in P$, $v \notin M$. This contradicts that M is a maximal clique. Hence, $P_f = \phi$.

   b) Show $X_f = \phi$: Assume $X_f \neq \phi \Rightarrow \exists x \in X_f$. This means x is adjacent to every $\omega_i \in M$. This implies $M \cup \{x\}$ is a valid clique. Since $x \in X$, $x \notin M$. This is a direct contradiction, as we found a clique $M \cup \{x\}$ which is strictly larger than the maximal clique M. Therefore, $X_f = \phi$.

   Since $P_f = \phi$ and $X_f = \phi$, the algorithm reports $R_f = M$.

4. **Show no duplicates are reported.**
   Assume for contradiction a maximal clique M is reported twice. This means there are two different call paths that result in $R = M, P = \phi, X = \phi$. Let the two paths diverge at a call BK(R, P, X). The algorithm iterates for v in P. Let the (arbitrary) order of vertices in P be $(v_1, v_2, \ldots, v_m)$.

   Path A: M is found in the recursive call starting with v_i: BK($R \cup \{v_i\}, P \cap n(v_i), X \cap n(v_i)$). This implies $v_i \in M$.

   Path B: M is found in the recursive call starting with v_j, where $j > i$. When this call is made, v_i has already been processed (in the i-th loop) and moved to the X set.

   Since Path A finds M, $v_i \in M$. Since Path B finds M, $v_j \in M$. Since M is a clique, v_i and v_j must be adjacent, so $v_i \in n(v_j)$. Since $i < j$, $v_i \in \{v_1, \ldots, v_{j-1}\}$. Because v_i is in this set and $v_i \in n(v_j)$, we know for certain that $v_i \in X_j$.

   In Path B, the algorithm continues recursively, adding all other vertices of M, let's call them $W = M \setminus (R \cup \{v_j\})$. The final X set when M is reported is $X_f = X_j \cap n(w_1) \cap n(w_2) \cap \ldots$ for all $w \in W$.

   We know $v_i \in X_j$. We also know $v_i \in M$ (from Path A) and all $w \in W$ are in M. Since M is a clique, v_i is adjacent to all $w \in W$. This means $v_i \in n(w)$ for all $w \in W$.

   Therefore, v_i must be in the final X set: $v_i \in X_f$. This means $X_f \neq \phi$.

   This is a contradiction. For M to be reported, the condition is $X_f = \phi$. Our assumption that M could be found in two different branches (Path A and Path B) leads to $X_f \neq \phi$, which prevents the report. Therefore, M can only be reported once.

$\square$

II. Bron-Kerbosch with Pivot

The "With Pivot" variant introduces a general strategy to reduce the branching factor of the algorithm by skipping redundant recursive calls.

Algorithm

```
procedure BK_Pivot(R, P, X)
    if P and X are empty then
        report R as a maximal clique
    end if
    Choose a pivot $u \in P \cup X$.
    for $v \in P \setminus n(u)$ do
        BK_Pivot($R \cup \{v\}$, $P \cap n(v)$, $X \cap n(v)$)
        $P \leftarrow P \setminus \{v\}$
        $X \leftarrow X \cup \{v\}$
    end for
end procedure
```

Correctness

Claim 2. BK_Pivot(g)enerates the same set of maximal cliques as the basic algorithm, regardless of which pivot u is chosen.

Proof. We must show that restricting the loop to $P \setminus n(u)$ does not miss any maximal cliques.

Let M be a maximal clique extending R (i.e., $R \subseteq M$). Consider the chosen pivot $u \in P \cup X$. There are two cases:

Case 1: $u \in M$
If $u \in M$, then u cannot be in X (otherwise M would have been found in a prior branch). Thus $u \in P$. Since u is not adjacent to itself, $u \notin n(u)$, so $u \in P \setminus n(u)$. The algorithm will select u, and M will be found.

Case 2: $u \notin M$
If $u \notin M$, then M cannot consist *only* of neighbors of u. If $M \subseteq n(u)$, then $M \cup \{u\}$ would be a valid clique (since u connects to all of M and all of R). This would imply M is not maximal. Therefore, there must be at least one vertex $v \in M$ such that v is **not** connected to u. $\Rightarrow \exists v \in M$ s.t. $v \in P \setminus n(u)$. The algorithm will select this v, and M will be found in that branch. $\square$

III. Tomita's Algorithm

Tomita's algorithm is the specific instance of the Pivot algorithm where the pivot is chosen optimally to minimize the worst-case running time.

Algorithm

```
1: procedure TOMITA(R, P, X)
2:     if P and X are empty then
3:         report R as a maximal clique
4:     end if
5:     Choose a pivot $u \in P \cup X$ such that $|P \cap n(u)|$ is maximized.
6:     for $v \in P \setminus n(u)$ do
7:         TOMITA($R \cup \{v\}$, $P \cap n(v)$, $X \cap n(v)$)
8:         $P \leftarrow P \setminus \{v\}$
9:         $X \leftarrow X \cup \{v\}$
10:    end for
11: end procedure
```

IV. Complexity Analysis (Tomita's Algorithm)

Claim 3 (Complexity)**.** The worst-case time complexity of Tomita's algorithm is $O(3^{n/3})$.

Proof. We must first show that the time complexity $T(n)$ is essentially proportional to the number of maximal cliques $M(n)$.

1. Relating Time to Clique Count
Let $T(n)$ be the time to process a graph of size n, and $M(n)$ be the number of maximal cliques. The recurrence for time is:

$$T(n) = \sum_{v \in P \setminus n(u)} T(|P \cap n(v)|) + O(n^2)$$

The recurrence for counting cliques is:

$$M(n) = \sum_{v \in P \setminus n(u)} M(|P \cap n(v)|)$$

In the worst-case scenario (exponential growth), the polynomial term $O(n^2)$ is negligible. Furthermore, in the worst-case graph (as derived below), there are no "dead ends" (branches that yield no cliques). Every branch leads to a distinct maximal clique. $\Rightarrow T(n) \approx M(n)$. Thus, we can bound the time complexity by finding the graph structure that maximizes $M(n)$.

2. The "Enemy" Graph Structure (Moon-Moser Theorem)
Let u, v be two non-adjacent vertices in a graph G. Let c_u be the number of maximal cliques containing u, and c_v be the number of maximal cliques containing v. *For contradiction*, assume the graph G is optimal (maximizes cliques) but $c_u > c_v$.

Construct a new graph G' by deleting v and adding a clone of u (call it u'). u' has the exact same neighbors as u. $\Rightarrow$ We lose c_v cliques (those involving v). $\Rightarrow$ But we gain c_u cliques (forming new cliques with u'). $\Rightarrow M(G') = M(G) - c_v + c_u$. This contradicts the optimality of G.

$\Rightarrow$ In the worst-case graph, all non-adjacent vertices must be "clones" (have identical neighborhoods). $\Rightarrow$ This structure is a **Complete Multipartite Graph** (disjoint clusters of independent sets).

3. Optimization of Cluster Size
Let the graph be divided into k clusters of size x, s.t. $n = k \cdot x$. Why is the number of cliques the product of sizes?

- **Constraint:** Inside a cluster (independent set), we can pick at most one vertex.
- **Maximality:** We must pick exactly one from each cluster to be maximal.
- **Independence:** The choices are independent across clusters.

By the Fundamental Counting Principle, the total number of cliques is the product: $\Rightarrow M(n) = \underbrace{x \cdot x \cdot \ldots \cdot x}_{k \text{ times}} = x^k = x^{n/x} = (x^{1/x})^n$.

We want to maximize this quantity. Since n is a constant, maximizing $(x^{1/x})^n$ is equivalent to maximizing the base $x^{1/x}$. Taking the derivative: $\frac{d}{dx}(x^{1/x}) = x^{1/x} \cdot \frac{1-\ln x}{x^2}$. Setting to 0: $1 - \ln x = 0 \Rightarrow x = e \approx 2.718$.

Since cluster sizes must be integers, we test the closest integers $x = 2$ and $x = 3$: $2^{1/2} \approx 1.414$ and $3^{1/3} \approx 1.442$. Since $3^{1/3} > 2^{1/2}$, the worst case occurs when the graph is composed of disjoint clusters of size 3. $\Rightarrow M(n) = 3^{n/3}$. □

V. Complexity Analysis (Basic Algorithm)

Claim 4 (Complexity)**.** The worst-case time complexity of the Basic Bron-Kerbosch algorithm (without pivoting) is $\Theta(2^n)$.

Proof. We analyze the behavior of the algorithm on a Complete Graph K_n. In K_n, every pair of vertices is connected, meaning $\forall v, n(v) = V \setminus \{v\}$.

1. Deriving the Recurrence Relation
Let $T(n)$ denote the number of recursive calls made by the algorithm when the candidate set P has size n (and X does not inhibit any choices).

Consider the loop for v in P inside a call with $|P| = n$:

- In the 1st iteration, we pick v_1. The recursive call receives $P' = P \cap n(v_1)$. Since the graph is complete, P' contains all vertices in P except v_1. Thus, the sub-problem size is $n-1$.
- After the return, v_1 is removed from P and added to X. The set P now has size $n-1$.
- In the 2nd iteration, we pick v_2. The recursive call receives $P' = (P \setminus \{v_1\}) \cap n(v_2)$. The sub-problem size is $n-2$.
- In general, for the k-th iteration ($1 \leq k \leq n$), the algorithm makes a recursive call with a candidate set of size $n-k$.

Including the cost of the current function call (denoted as 1), the total number of calls satisfies the recurrence:

$$T(n) = 1 + \sum_{k=1}^{n} T(n-k) = 1 + \sum_{j=0}^{n-1} T(j)$$

where $T(0) = 1$ (the base case where P is empty and the algorithm reports R).

2. Solving by Induction
We claim that $T(n) = 2^n$.

Base Case:
For $n = 0$, $T(0) = 1$. The formula holds.

Inductive Hypothesis:
Assume $T(k) = 2^k$ for all $k < n$.

Inductive Step:
Substitute the hypothesis into the recurrence:

$$T(n) = 1 + \sum_{j=0}^{n-1} 2^j$$

The term $\sum_{j=0}^{n-1} 2^j$ is a geometric series with sum $2^n - 1$.

$$T(n) = 1 + (2^n - 1)$$

$$T(n) = 2^n$$

3. Conclusion
Since the algorithm performs exactly 2^n calls on a complete graph K_n, the worst-case time complexity is $\Theta(2^n)$. This confirms that without the pivot optimization, the algorithm enumerates all cliques (every subset of vertices in K_n) rather than just maximal cliques. □

VI. Complexity Analysis (Pivot Algorithm)

Claim 5 (Complexity)**.** The worst-case time complexity of the Bron-Kerbosch algorithm with an arbitrary Pivot is $\Theta(3^{n/3})$.

Proof. To prove the worst-case bound, we construct a graph G that maximizes the number of recursive steps and solve the resulting recurrence relation.

1. The Worst-Case Graph Structure (M_n)
Let M_n be a complete multipartite graph consisting of $k = n/3$ disjoint sets of vertices $\{S_1, S_2, \ldots, S_k\}$, where each set S_i has size 3 (i.e., $|S_i| = 3$).

- **Inside a set:** Vertices within the same S_i are *not* connected (independent sets).
- **Between sets:** Every vertex in S_i is connected to every vertex in S_j (for $i \neq j$).

2. Deriving the Recurrence Relation
Let $T(n)$ be the number of recursive calls on a graph of this structure with n vertices. In the call BK_PIVOT(R, P, X):

1. **Pivot Selection:** The algorithm picks a pivot $u \in P$. Due to the symmetric nature of M_n, let us assume u belongs to the set S_1.
2. **Determine Loop Range:** The loop iterates over $v \in P \setminus n(u)$. In M_n, the neighbors of u, $n(u)$, are exactly all vertices in $S_2 \cup S_3 \cup \cdots \cup S_k$. Therefore, the non-neighbors of u are exactly the vertices in its own set S_1.

   $$P \setminus n(u) = S_1$$

   Since $|S_1| = 3$, the loop will execute exactly 3 times.
3. **Recursive Step:** For each $v \in S_1$:

   $$P_{new} = P \cap n(v)$$

   Since v connects to all sets except S_1, the size of the sub-problem is $|P_{new}| = n - 3$.

Thus, the recurrence relation is:

$$T(n) = \sum_{v \in S_1} T(n-3) = 3 \cdot T(n-3)$$

3. Solving by Induction
We claim $T(n) = 3^{n/3}$.

Base Case: For $n = 0$, $T(0) = 1$ (reporting the clique). $3^0 = 1$. Holds.

Inductive Hypothesis: Assume $T(k) = 3^{k/3}$ for $k < n$.

Inductive Step:

$$T(n) = 3 \cdot T(n-3)$$

Substitute hypothesis:

$$T(n) = 3 \cdot 3^{(n-3)/3}$$

$$T(n) = 3^1 \cdot 3^{n/3-1}$$

$$T(n) = 3^{n/3}$$

4. Conclusion
On the graph M_n, the Pivot algorithm performs $3^{n/3}$ operations. Since this matches the lower bound required to list the maximal cliques (of which there are $3^{n/3}$ in M_n), the worst-case complexity is exactly $\Theta(3^{n/3})$. $\square$

7 Proof 1: Correctness of the ILP Formulation

7.1 Problem Definitions

Let $G = (V, E)$ be a graph. A subset of vertices $S \subseteq V$ is an **Independent Set** if no two vertices in S are connected by an edge:

$$\forall u, v \in S, \quad (u, v) \notin E$$

We formulate this as an Integer Linear Program (ILP) with binary variables $x_i \in \{0, 1\}$:

$$\text{Maximize } Z = \sum_{i \in V} x_i \tag{1}$$

$$\text{Subject to } x_u + x_v \leq 1 \quad \forall (u, v) \in E \tag{2}$$

$$x_i \in \{0, 1\} \quad \forall i \in V \tag{3}$$

Theorem 1. The optimal solution to the ILP formulation corresponds exactly to the Maximum Independent Set of graph G.

Proof. We prove this by establishing a bijection between the set of feasible binary vectors $\mathbf{x}$ and the set of valid independent sets S.

1. Feasibility (ILP $\implies$ Graph) Let $\mathbf{x}$ be any feasible solution to the ILP. We construct a set $S_{\mathbf{x}} = \{i \in V \mid x_i = 1\}$. Suppose $S_{\mathbf{x}}$ is *not* an independent set. Then there must exist two vertices $u, v \in S_{\mathbf{x}}$ such that $(u, v) \in E$. If $u, v \in S_{\mathbf{x}}$, then $x_u = 1$ and $x_v = 1$. Substituting this into the edge constraint (2):

$$x_u + x_v = 1 + 1 = 2$$

However, the constraint requires $x_u + x_v \leq 1$. This is a contradiction. Thus, $S_{\mathbf{x}}$ must be a valid independent set.

2. Completeness (Graph $\implies$ ILP) Let S be any valid independent set. We define vector $\mathbf{x}$ such that $x_i = 1$ if $i \in S$, else 0. For any edge $(u, v) \in E$, it is impossible for both u and v to be in S (by definition of independence). Thus, at least one variable is 0. The sum $x_u + x_v$ can be 0 or 1, but never 2. Therefore, $x_u + x_v \leq 1$ is always satisfied.

3. Optimality The objective function is $Z = \sum x_i$. This sum is exactly equal to the cardinality $|S|$. Maximizing the sum is mathematically equivalent to maximizing the size of the set. □

8 Proof 2: The LP Relaxation is an Upper Bound

8.1 Definitions

Let Z_{ILP}^* be the optimal value of the Integer Linear Program defined above. Let Z_{LP}^* be the optimal value of the **Linear Programming Relaxation**, where the integrality constraint $x_i \in \{0,1\}$ is replaced by $0 \leq x_i \leq 1$.
Theorem 2. The optimal value of the Linear Programming relaxation is greater than or equal to the optimal value of the original Integer Linear Program:

$$Z_{LP}^* \geq Z_{ILP}^*$$

Proof. The proof relies on the relationship between the feasible regions of the two optimization problems.

1. Subset Relationship Let $\mathcal{F}ILP$ be the set of all feasible solutions for the ILP, and $\mathcal{F}LP$ be the set of feasible solutions for the LP. Any solution $\mathbf{x} \in \mathcal{F}_{ILP}$ must satisfy $x_i \in \{0,1\}$. Since $0 \leq 0 \leq 1$ and $0 \leq 1 \leq 1$, any valid binary value automatically satisfies the continuous constraint $0 \leq x_i \leq 1$. Therefore, every feasible solution to the ILP is also a feasible solution to the LP:

$$\mathcal{F}ILP \subseteq \mathcal{F}LP \tag{4}$$

2. Maximization Property Both problems maximize the same objective function $f(\mathbf{x}) = \sum x_i$. It is a fundamental property of optimization that the maximum value of a function over a set is less than or equal to the maximum value over any superset. Since $\mathcal{F}ILP \subseteq \mathcal{F}LP$:

$$\max_{\mathbf{x}\in\mathcal{F}ILP} f(\mathbf{x}) \leq \max \mathbf{x} \in \mathcal{F}_{LP} f(\mathbf{x}) \tag{5}$$

Conclusion By definition, $Z_{ILP}^* = \max_{\mathbf{x}\in\mathcal{F}ILP} f(\mathbf{x})$ and $ZLP^* = \max_{\mathbf{x}\in\mathcal{F}_{LP}} f(\mathbf{x})$. Substituting these into the inequality yields:

$$Z_{ILP}^* \leq Z_{LP}^*$$

Thus, the LP solution provides a mathematical upper bound on the maximum clique size. $\square$

The Geometry of Hardness
From Algorithms to the Overlap Gap Property Formal Theory & Intuitive Understanding December 1, 2025

9 Introduction

This document explores why finding the Maximum Clique in random graphs is computationally intractable. We link the failure of backtracking algorithms (like Bron-Kerbosch and Tomita) to a topological feature of the solution space known as the *Overlap Gap Property* (OGP).

10 Given: Random Graph

Definition 4 (Erdős-Rényi Graph $G(n,p)$)**.** Let V be a set of n vertices. The random graph $G(n,p)$ is constructed by including every possible edge (u,v) independently with probability p. For the "hard" regime, we consider the dense case where $p = 1/2$.

Intuitive Explanation

We basically flip a coin (run a Bernoulli trial) for each pair of nodes of the graph(there are n nodes) and the result determines if there is an edge connecting the nodes/vertices or not. So in $G(n,p)$, n is the number of nodes and p is the bias of the coin flipped. Could also be thought of as there are n people and coin flip for each pair of people determines if they know each other or not.

11 Goal: Maximum Clique

Definition 5 (Clique)**.** This we can refer to the previous part

Intuitive Explanation

We are looking for the biggest group of people(in terms of number) in the room where **everyone knows everyone else**. As we shall see, In a random graph, small-sized friend groups are everywhere and huge friend groups are extremely rare.

12 Exploring the graph for goal: Tomita's Algorithm

To see why the algorithm fails, we see what it actually does:
Definition 6 (Pivot Optimization)**.** Let P be the set of candidate vertices and X be the set of excluded vertices. A pivot vertex $u \in P \cup X$ is chosen to prune the search tree. The algorithm only recursively explores vertices in the set:

$$P \setminus N(u)$$

where $N(u)$ is the set of neighbors of u.

Intuitive Explanation

Working of the Pivot: If you are looking for a clique and you decide to include Person A (the pivot), you don't need to start a *new* search with Person A's friends later. Why? Because if a clique contains Person A, your current search will find it. If a clique contains only Person A's friends (but not Person A), you can find it by just adding Person A to it later. Therefore, the Pivot allows you to skip checking the neighbors of the pivot node.

13 The Barrier: The Overlap Gap Property (OGP)

This is the core mathematical reason for hardness. It describes the geometry of the "Solution Space."

Definition 7 (Overlap)**.** Let S_1 and S_2 be two distinct maximal cliques of size near $2\log_2 n$. The **overlap** is defined as the number of shared vertices (intersection):

$$\mathcal{O}(S_1, S_2) = |S_1 \cap S_2|$$

Theorem 3 (The Overlap Gap Property)**.** For a random graph $G(n, 1/2)$, there exist two constants $0 < \nu_1 < \nu_2 < 1$ such that with high probability, for any two large cliques S_1, S_2:

$$\frac{|S_1 \cap S_2|}{|S_1|} \in [0, \nu_1] \cup [\nu_2, 1]$$

The interval (ν_1, ν_2) is a **forbidden region**.

Intuitive Explanation

- **Indistinguishability:** Two large cliques are either almost identical (sharing $> 90\%$ of vertices).
- **Distinguishability:** Or they are completely distinct (sharing $< 10\%$ of vertices).
- **Gap:** There is no middle ground. You will **never** find two cliques that are "somewhat similar" (e.g., sharing 50%).

This means that the solution space is not a single component. It is rather broken into thousands of tiny, isolated components with no connections between them.
Note: This "Forbidden Region" phenomenon is a rigorous result in probabilistic combinatorics, notably analyzed by D. Gamarnik (e.g., PNAS 2021, "The Overlap Gap Property").

14 The Failure: Why OGP Breaks the Pivot

Lemma 1 (Ineffectiveness of Constraints)**.** Let $u \in C_A$ be the pivot vertex. Let C_B be a disjoint maximal clique. By the Overlap Gap Property, $|C_A \cap C_B| \leq \nu_1 |C_A|$. This implies that u is unlikely to have edges connecting to C_B (due to the random edge distribution between disjoint sets).

$$P \setminus N(u) \supseteq C_B \setminus \{\text{small noise}\}$$

Thus, the pruning step fails to eliminate the search space for C_B.

Intuitive Explanation

The Failure of Information Propagation: Tomita's pivot is designed to propagate constraints: "If I pick this vertex, I implicitly handle its neighbors." This relies on the solutions being structurally connected.
However, OGP proves that optimal solutions in random graphs are **statistically decoupled**. Choosing a pivot in one solution gives virtually zero information about the existence or location of other solutions. Because the constraints (edges) do not propagate across the gap between clusters, the algorithm cannot "prune" its way from one solution to another; it is mathematically forced to treat them as independent, brute-force search problems.

15 Brief summary of the rigorous result in PNAS 2021

How do we know this "Gap" actually exists? The proof uses the **First Moment Method** to analyze the expected number of pairs of solutions.

15.1 Step 1: The Counter

We define a random variable Z_z that counts the number of pairs of cliques (σ_1, σ_2) that share exactly z vertices.

$$Z_z = \sum_{\sigma_1,\sigma_2} \mathbb{I}\left(\sigma_1, \sigma_2 \in \mathcal{C}_k \text{ and } |\sigma_1 \cap \sigma_2| = z\right)$$

15.2 Step 2: The Expectation (The U-Curve)

We calculate $\mathbb{E}[Z_z]$ by balancing two opposing forces:

$$\mathbb{E}[Z_z] = \underbrace{\binom{n}{k}\binom{k}{z}\binom{n-k}{k-z}}_{\text{Entropy (Ways to pick sets)}} \times \underbrace{p^{\text{edges}}}_{\text{Cost (Probability of edges)}}$$

Intuitive Explanation

The Balance:

- **Entropy (Pushing Up):** There are exponentially many ways to pick two groups of people. This tries to make the count of pairs huge.
- **Probability Cost (Pushing Down):** Every person added to the group requires specific edges to exist. This probability drops exponentially fast.

15.3 Step 3: The Conclusion

When we plot $\mathbb{E}[Z_z]$ against overlap z, it forms a "U-shape":

- At **Ends** ($z \approx 0$ or $z \approx k$), the Entropy wins. Pairs exist.
- At the **Middle** ($z \approx k/2$), the Cost overwhelms the Entropy. The expected number of pairs drops well below 1.

By Markov's Inequality, if $\mathbb{E}[Z_z] \to 0$, then the probability of such pairs existing is zero. This mathematically proves the existence of the forbidden region.

16 Conclusion

The computational hardness of finding the Maximum Clique is not due to a lack of clever algorithms. It is a structural inevitability caused by the **Overlap Gap Property**.

Because the optimal solutions are topologically isolated (shattered) with no intermediate overlap, no local information (like pivoting) can guide the algorithm from one solution to another. The algorithm is forced to traverse the entire exponentially large space, visiting each ”island” individually.