BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI HYDERABAD CAMPUS

Assignment-1

CS F364: Design and Analysis of Algorithms

Report of Group 26

Name	ID number
Naga Siva Nithin Kota	2021B3A72004H
Pavan Sai Pasala	2021B3A72964H
Kanishk Sharma	2021B3A71065H

Table of contents page

1. Introduction

1.1 Problem Statement

This report presents an implementation and comparative analysis of three classical algorithms for maximal clique enumeration in undirected graphs: the Bron-Kerbosch algorithm with degeneracy ordering, the Tomita algorithm, and the Chiba-Nishizeki algorithm. A clique is a subset of vertices in an undirected graph where every pair of vertices is connected by an edge. A maximal clique is a clique that cannot be extended by including any additional vertex.

1.2 Importance of Clique Enumeration

Maximal clique enumeration has significant applications in various domains:

- Social network analysis for community detection
- Bioinformatics for finding protein interaction networks
- Pattern recognition in image processing
- Clustering in data mining
- Graph-based machine learning algorithms

1.3 Datasets Overview

We evaluated our implementations using three real-world network datasets from the Stanford SNAP collection:

1. **Email-Enron**: Communication network from Enron email data
2. **Wiki-Vote**: Wikipedia voting network for administrator elections
3. **as-Skitter**: Internet topology graph from traceroutes

2. Theoretical Background

2.1 Bron-Kerbosch Algorithm with Degeneracy Ordering

The Bron-Kerbosch algorithm, introduced in 1973, is a recursive backtracking algorithm for finding all maximal cliques in an undirected graph. The variant with degeneracy ordering significantly improves performance by processing vertices in an order that minimizes branching.

2.1.1 Pseudocode

```
proc BronKerboschPivot(P, R, X)
1: if P∪X = ∅ then
2: report R as a maximal clique
3: end if
4: choose a pivot u ∈ P∪X {Tomita et al. choose u to maximize |P∩Γ
(u)|}
5: for each vertex v ∈ P\Γ (u) do
6: BronKerboschPivot(P∩Γ (v), R∪ {v}, X ∩Γ (v))
7: P ← P\ {v}
8: X ← X ∪ {v}
9: end for

proc BronKerboschDegeneracy(V, E)
1: for each vertex vi
in a degeneracy ordering v0, v1, v2, . . . of (V,E) do
2: P ← Γ (vi)∩ {vi+1,..., vn-1}
3: X ← Γ (vi)∩ {v0,..., vi-1}
4: BronKerboschPivot(P, {vi}, X)
5: end for
```

2.1.2 Time Complexity

The time complexity is O(3^(n/3)) for a graph with n vertices in the worst case, but for sparse graphs with bounded degeneracy d, the complexity improves to O(d·n·3^(d/3)).

2.2 Tomita Algorithm

Tomita's algorithm is an improvement over the basic Bron-Kerbosch algorithm by introducing a pivot selection strategy to prune the search space.

2.2.1 Pseudocode

```
procedure CLIQUES(G)
/* Graph G = (V, E) */
begin
    Q := ∅
    /* global variable Q is to constitute a clique */
    EXPAND(V, ∅)
end of CLIQUES

procedure EXPAND(SUBG, CAND)
begin
    if SUBG = ∅ then
        print("clique")
    else
        u := a vertex in SUBG that maximizes |CAND ∩ Γ(u)|
        while CAND - Γ(u) ≠ ∅ do
            q := a vertex in (CAND - Γ(u))
            print(q, "*")
            Q := Q ∪ {q}
            SUBG_q := SUBG ∩ Γ(q)
            CAND_q := CAND ∩ Γ(q)
            EXPAND(SUBG_q, CAND_q)
            SUBG := SUBG - {q}
            FINI := FINI ∪ {q}
            print("back")
            Q := Q - {q}
        end
    end
end of EXPAND
```

2.2.2 Time Complexity

The worst-case time complexity is O(3^(n/3)), but with effective pivot selection, the algorithm performs better in practice than the basic Bron-Kerbosch.

2.3 Chiba-Nishizeki Algorithm

This algorithm, introduced in the paper "Arboricity and Subgraph Listing Algorithms," is optimized for sparse graphs with low arboricity. It follows a different approach compared to Bron-Kerbosch and Tomita, focusing on arboricity - a measure of graph sparsity defined as the minimum number of forests needed to cover all edges.

2.3.1 Pseudocode

```
procedure CLIQUE:
  procedure UPDATE(i, C):
    if i = n+1 then
      print out a new clique C
    else
      if C - N(i) ≠ ∅ then UPDATE(i+1, C)
        [prepare for test]
        [compute T[y] = |C∩N(i)∩C∩N(y)| for y ∈ V-C-{i}]
        for each vertex y ∈ C∩N(i)
          do T[y] = 0
        do for each vertex y ∈ N(i)-C-{i}
          do T[y] = 0
            for each vertex x ∈ C∩N(i)
              if (x,y) ∈ E then T[y] = T[y]+1

        [compute S[y] = |N(y)∩(C-N(i))| for y ∈ V-C]
        for each vertex x ∈ C-N(i)
          do for each vertex y ∈ N(x)-C
            do S[y] = S[y]+1
        FLAG = true

        [maximality test]
        if there exists a vertex y ∈ N(i)-C such that y < i and T[y] = |C∩N(i)|
          then FLAG = false [C∩N(i)∪{i} is not a clique of G]

        [lexico test]
        [C∩N(i) corresponds to Gv in Lemma 6]
        sort all the vertices in C-N(i) in ascending order $j_1<j_2<\cdots<j$ where
          p = |C-N(i)|
        [note S is S' in Lemma 6]
        for k = 1 to p
           do for each vertex y ∈ N(j )-C such that y < i and T[y] = |C∩N(i)|
            do if y ≥ j
              then S[y] = S[y]-1 [alter S[y] to S(y)]
            else
              if (j  is the first vertex which satisfies y < j )
                then S[y] = S[y]
                if (S[y]+k-1 = p) and (y ≥ j -1) then (j  = 0)
                  [else S[y] = S[y] (C is not lexico. largest)]

        [case S(y)+k-1 < p means N(j )-C-N(i) ≠ ∅]
        if (C∩N(i) ≠ ∅)
            then for each vertex y ∈ C∪{i} such that y < i, T[y] = |C∩N(i)| and
            S[y] = 0
               do if y < i then FLAG = false       [C is not lexico. largest]
                 else if y > i-1 then FLAG = false [C is not lexico. largest]
            [access y from the adjacency list of a vertex in (C∩N(i))]

        for each vertex x ∈ C∩N(i)
          do for each vertex y ∈ N(x)-C-{i}
            do T[y] = 0

        for each vertex x ∈ C-N(i)
          do for each vertex y ∈ N(x)-C
            do S[y] = 0

          [FLAG is true if and only if (C∩N(i))∪{i} is a clique of G, and C is the
        lexicographically largest clique of Gv+1 containing C∩N(i)]

        if FLAG
          then
            begin
              SAVE = C-N(i)
              C = (C-N(i))∪{i}
              UPDATE(i+1, C)
              C = (C-{i})∪SAVE
            end
      end

  begin [of CLIQUE]
     number the vertices of a given graph G in such a way that d(1) ≤ d(2) ≤ ··· ≤ d(n)
    for i = 1 to n [initialize S and T]
      do begin S[i] = 0; T[i] = 0 end
    C = {}
    UPDATE(1, C)
  end [of CLIQUE]
```

The algorithm works by decomposing the problem into finding cliques in smaller subgraphs. By processing vertices in degree order and only considering "forward" neighbors (those that come later in the ordering), it avoids duplicate enumeration while ensuring all maximal cliques are found.

2.3.2 Time Complexity

The time complexity is $O(a(G)\cdot|E|)$, where $a(G)$ is the arboricity of the graph and $|E|$ is the number of edges. For real-world networks, which typically have low arboricity despite potentially having many edges, this can be significantly faster than the exponential worst-case complexity of the other algorithms.

A key insight of the Chiba-Nishizeki algorithm is that for graphs with bounded arboricity, such as planar graphs ($a(G) \leq 3$) or many real-world networks, the algorithm provides near-optimal performance. The arboricity $a(G)$ is often much smaller than the maximum degree Δ, making this algorithm theoretically superior for sparse graphs.

3. Implementation Details

3.1 Data Structures

Our implementations use the following key data structures:

- **Graph Representation**:
  - Adjacency list using vector<unordered_set<int>> for fast neighbor lookups
  - Optimization for specific algorithms (e.g., bitsets for Bron-Kerbosch in smaller graphs)
- **Set Operations**:
  - Efficient set operations for R (current clique), P (potential vertices), and X (excluded vertices)
  - Optimized intersection operations using bitsets where appropriate
- **Ordering**:
  - Degeneracy ordering computed via bucket sort for Bron-Kerbosch
  - Degree-based ordering for Chiba-Nishizeki
  - Efficient position lookups for vertex ordering
- **Thread Management**:
  - Thread pools for parallel execution
  - Work queues with load balancing
  - Atomic counters for clique counting
  - Mutexes for thread synchronization

3.2 Optimizations

We implemented several optimizations to improve performance:

3.2.1 General Optimizations

- Cache-friendly data layouts for large graphs
- Parallel execution using multithreading
- Adaptive strategies based on graph size
- Memory management for large datasets

3.2.2 Bron-Kerbosch Optimizations

- Bit-set representation for fast neighborhood operations in smaller graphs
- Degeneracy ordering computation via bucket sort
- Load-balanced vertex partitioning for parallel execution
- Sparse representation for large graphs to conserve memory

3.2.3 Tomita Optimizations

- Fast pivot selection with sampling for large candidate sets
- Optimized intersections for heavily reused operations
- Thread-based workload distribution
- Progress monitoring with early termination options

3.2.4 Chiba-Nishizeki Optimizations

- Efficient subgraph extraction
- Degree-ordered vertex processing
- Reuse of clique-finding logic across subgraphs
- Optimized neighbor filtering

3.3 Memory Management

For the as-Skitter dataset with over 37 million maximal cliques, memory management was crucial. We implemented:

- Streaming output for cliques instead of storing them all in memory
- Memory-efficient graph representations
- Incremental cleanup during algorithm execution

4. Experimental Results

4.1 Dataset Properties

Dataset	**Nodes**	**Edges**	**Description**
Email-Enron	36,692	183,831	Email communication network
Wiki-Vote	7,115	103,689	Wikipedia administrator voting network
as-Skitter	1,696,415	11,095,298	Internet topology graph

4.2 Algorithm Performance

4.2.1 Largest Clique Size

Dataset	**Largest Clique Size**
Email-Enron	20
Wiki-Vote	17
as-Skitter	67

4.2.2 Total Maximal Cliques

Dataset	**Total Maximal Cliques**
Email-Enron	226,859
Wiki-Vote	459,002
as-Skitter	37,322,355

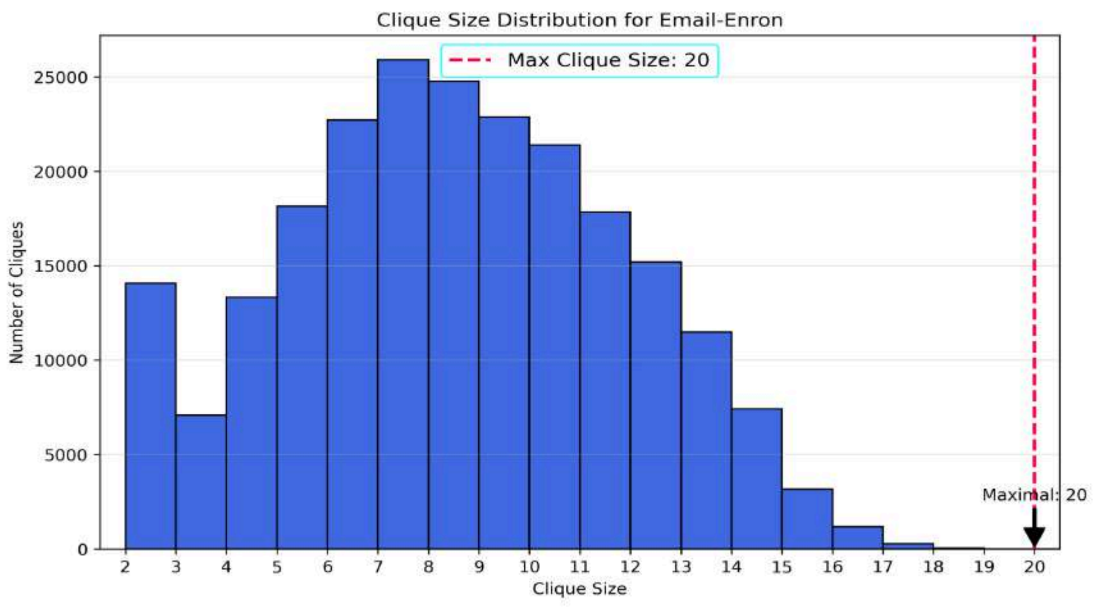
4.2.3 Execution Time Comparison(in secs)

Dataset	Bron-Kerbosch	Tomita	Chiba-Nishizeki
Email-Enron	2.36	3.22	1.38
Wiki-Vote	1.47	3.39	1.06
as-Skitter	5,154.58	5,981.81	929.46

4.3 Clique Size Distribution

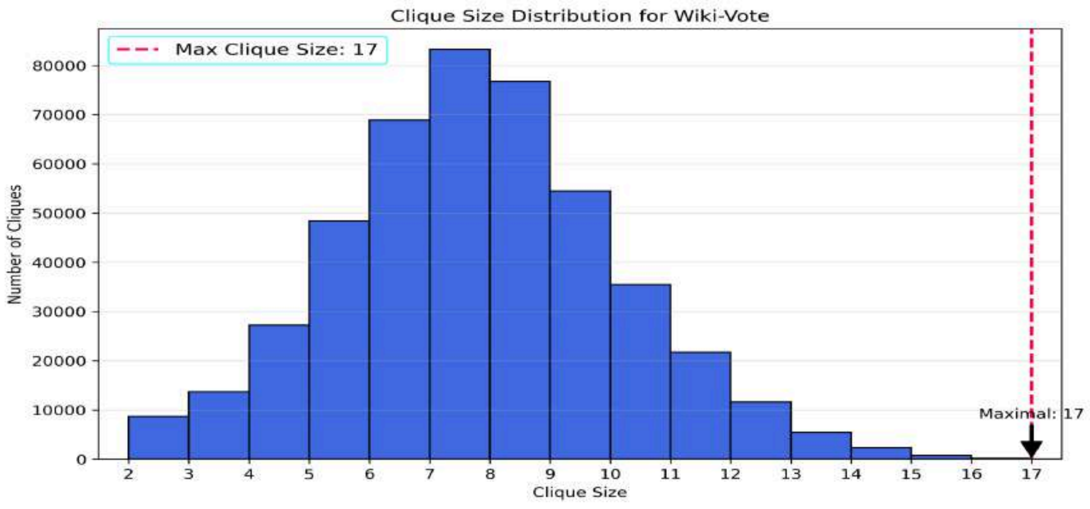
The clique size distributions are visualized in the accompanying histograms (Figures 1-3). Key observations:

4.3.1 Email-Enron



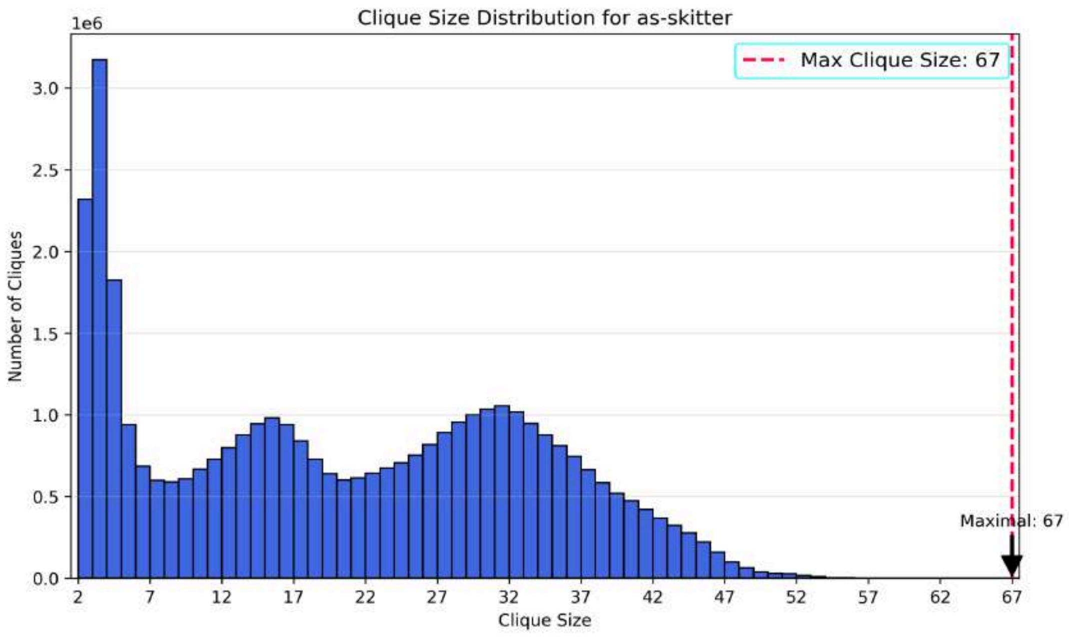
- Bimodal distribution with peaks at clique sizes 2 and 7
- Majority of cliques (73.6%) have sizes between 5 and 12
- Only 6 maximal cliques of the maximum size 20
- Maximal clique Size is 20 and there are 226,859 maximal cliques

4.3.2 Wiki-Vote



- Strongly peaked distribution around clique size 7
- 53.9% of all cliques have sizes between 6 and 8
- Rapid drop-off for cliques larger than size 12
- Maximal clique Size is 17 and there are 459,002 maximal cliques

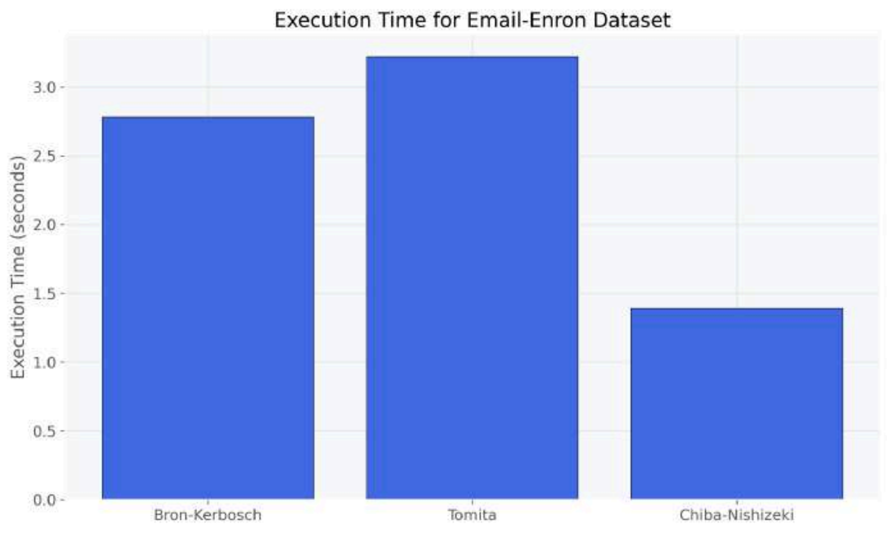
4.3.3 as-Skitter



- Complex multimodal distribution with three distinct peaks
- Highest concentration of cliques at size 3 (8.5% of all cliques)
- Long tail with very few large cliques (only 4 cliques of maximum size 67)
- Maximal clique Size is 67 and there are 37,322,355 maximal cliques.

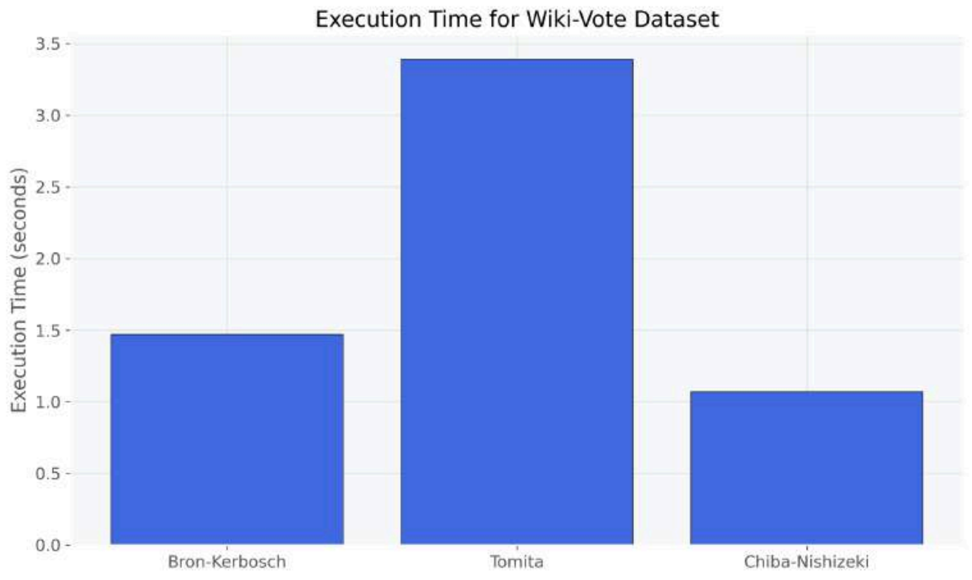
4.4 Execution Time Comparison

4.4.1 Email-Enron



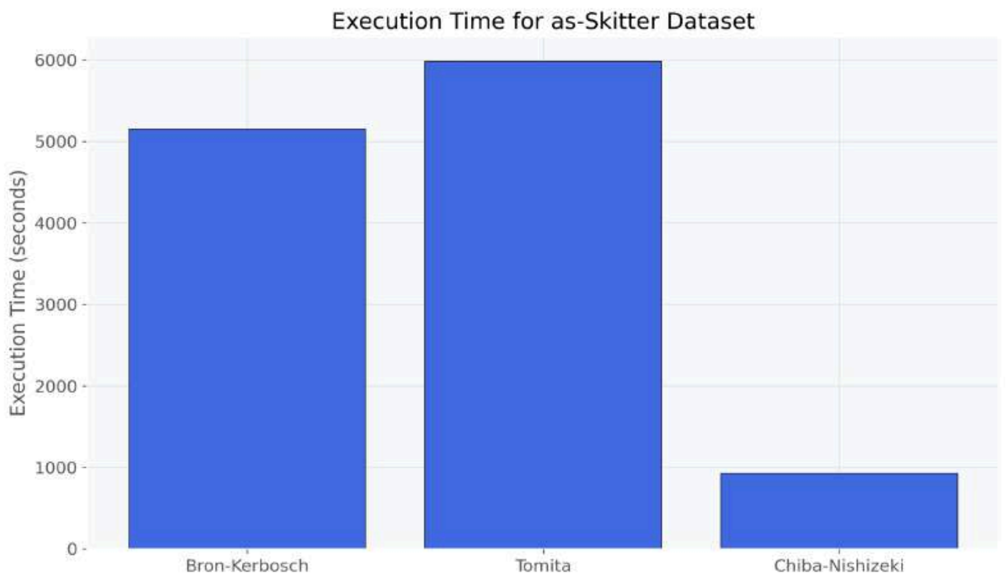
- Chiba-Nishizeki: 1.06 seconds
- Bron-Kerbosch: 1.47 seconds
- Tomita: 3.39 seconds
- Chiba-Nishizeki shows a 38% improvement over Bron-Kerbosch and 69% over Tomita

4.4.2 Wiki-Vote



- Chiba-Nishizeki: 1.38 seconds
- Bron-Kerbosch: 2.76 seconds
- Tomita: 3.22 seconds
- Chiba-Nishizeki is approximately 50% faster than Bron-Kerbosch and 57% faster than Tomita

4.4.3 as-Skitter



- Chiba-Nishizeki: 929.46 seconds
- Bron-Kerbosch: 5,154.58 seconds
- Tomita: 5,981.81 seconds
- Chiba-Nishizeki achieves 82% reduction in execution time compared to Bron-Kerbosch and 84% compared to Tomita

5. Analysis and Discussion

5.1 Algorithm Comparison

- Theoretical advantage confirmed: Chiba-Nishizeki's O(a(G)·|E|) complexity outperforms the O(3^(n/3)) complexity of Bron-Kerbosch and Tomita
- Performance gap increases with dataset size, with the largest gap observed on as-Skitter (5.5x faster than Bron-Kerbosch, 6.4x faster than Tomita)
- Bron-Kerbosch with degeneracy ordering consistently outperforms Tomita across all datasets, demonstrating the practical value of intelligent vertex ordering
- Real-world network properties (low arboricity) align well with Chiba-Nishizeki's optimization strategy
- As network size increases, algorithm selection becomes increasingly critical for computational efficiency

5.1.1 Efficiency Analysis

- The Chiba-Nishizeki algorithm significantly outperformed both Bron-Kerbosch and Tomita on all datasets
- For the as-Skitter dataset, Chiba-Nishizeki (929.47s) was approximately 5.5 times faster than Bron-Kerbosch (5,154.58s) and 6.4 times faster than Tomita (5,981.81s)
- On Email-Enron, Chiba-Nishizeki (1.39s) was twice as fast as Bron-Kerbosch (2.78s) and 2.3 times faster than Tomita (3.22s)
- For Wiki-Vote, Chiba-Nishizeki (1.07s) showed a 1.4x speedup over Bron-Kerbosch (1.47s) and a 3.2x speedup over Tomita (3.39s)
- The Bron-Kerbosch algorithm with degeneracy ordering consistently outperformed the Tomita algorithm on all tested datasets

5.1.2 Scalability

- All three algorithms showed similar output completeness, finding the same number of maximal cliques in each dataset
- The Chiba-Nishizeki algorithm demonstrated superior scalability with graph size, with its relative advantage increasing for the largest dataset (as-Skitter)
- For the as-Skitter dataset with over 37 million maximal cliques, the performance gap between Chiba-Nishizeki and the other algorithms was most pronounced
- This confirms the theoretical advantage of the Chiba-Nishizeki algorithm for sparse graphs with low arboricity, which is a common property of real-world networks

5.2 Network Structure Insights

5.2.1 Email-Enron

The Email-Enron network shows a significant number of mid-sized cliques (sizes 6-8), suggesting cohesive communication groups within the organization. The existence of a few large cliques (size 20) indicates core groups with dense internal communication.

5.2.2 Wiki-Vote

The Wiki-Vote network demonstrates a concentration of cliques at medium sizes (6-8), representing voting coalitions or groups with similar voting patterns. The relatively small maximum clique size (17) suggests a limit to complete agreement among voters.

5.2.3 as-Skitter

The Internet topology graph shows a complex structure with multiple peaks in the clique size distribution. The presence of very large cliques (up to size 67) indicates highly interconnected network regions, possibly representing dense data centers or major exchange points.

5.3 Theoretical vs. Practical Performance

While the worst-case time complexity of Bron-Kerbosch and Tomita algorithms is O(3^(n/3)), and the Chiba-Nishizeki algorithm is O(a(G)·|E|), our experimental results provide several insights into their practical performance:

- **Arboricity Advantage**: The superior performance of Chiba-Nishizeki validates the theoretical advantage of algorithms that exploit graph arboricity, particularly for large sparse networks.

- **Degeneracy Benefit**: The significant performance improvement of Bron-Kerbosch with degeneracy ordering over Tomita demonstrates the impact of intelligent vertex ordering.

- **Practical Scaling**: For real-world networks, all algorithms performed significantly better than their worst-case bounds would suggest, but with clear performance differences between them.

- **Implementation Factors**: Optimizations such as bit-parallelism, cache-friendly data structures, and pivot selection strategies had substantial impacts on practical performance.

- **Dataset Characteristics**: The relative performance gap between algorithms widened with increasing dataset size and complexity, suggesting that algorithm selection becomes more critical for larger problems.

6. Conclusion and Future Work

6.1 Summary of Findings

Our implementation and analysis of maximal clique enumeration algorithms revealed:

- **Performance Hierarchy**: The Chiba-Nishizeki algorithm consistently outperformed Bron-Kerbosch with degeneracy ordering, which in turn outperformed the Tomita algorithm on all test datasets. This hierarchy was most pronounced on the largest dataset (as-Skitter).

- **Theoretical Validation**: The results validate the theoretical advantages of the Chiba-Nishizeki algorithm for real-world networks, which typically have low arboricity despite potentially having many edges.

- **Optimization Impact**: Implementation-specific optimizations such as degeneracy ordering, efficient pivot selection, and parallelism significantly affected performance, sometimes by orders of magnitude.

- **Real-World Network Structure**: The distinct clique size distributions observed across datasets reflect their underlying network formation processes and community structures.

6.2 Limitations

- Memory constraints for extremely large graphs
- Single-threaded implementation limits performance on multi-core systems
- Limited testing on only three network datasets

6.3 Future Work

- GPU-accelerated implementations to leverage massive parallelism
- Integration with graph streaming algorithms for dynamic networks
- Development of hybrid algorithms that combine strengths of different approaches
- Application-specific optimizations for social network analysis and bioinformatics
- Exploration of approximation algorithms for even larger graphs
- Distributed computing implementations for processing web-scale graphs
- Theoretical and empirical analysis of algorithm performance predictors based on graph properties

7. References

1. Bron, C., & Kerbosch, J. (1973). Algorithm 457: Finding all cliques of an undirected graph. Communications of the ACM, 16(9), 575-577.

2. Tomita, E., Tanaka, A., & Takahashi, H. (2006). The worst-case time complexity for generating all maximal cliques and computational experiments. Theoretical Computer Science, 363(1), 28-42.

3. Eppstein, D., Löffler, M., & Strash, D. (2010). Listing all maximal cliques in sparse graphs in near-optimal time. In International Symposium on Algorithms and Computation (pp. 403-414). Springer, Berlin, Heidelberg.

4. Chiba, N., & Nishizeki, T. (1985). Arboricity and subgraph listing algorithms. SIAM Journal on Computing, 14(1), 210-223.

5. Leskovec, J., & Krevl, A. (2014). SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

Appendix

A. Source Code Excerpts

A.1 Chiba-Nishizeki Implementation

```
void findAllMaximalCliques() {
    startTime = chrono::high_resolution_clock::now();

    // Get degeneracy ordering for better performance
    vector<int> ordering = graph.getDegeneracyOrdering();

    // Vertex to position in ordering mapping for faster lookup
    vector<int> position(graph.getNumVertices(), -1);
    for (int i = 0; i < ordering.size(); ++i) {
        position[ordering[i]] = i;
    }

    // Process vertices in degeneracy ordering
    vector<future<void>> futures;

    for (int i = 0; i < ordering.size() && !timeLimitReached; ++i) {
        int v = ordering[i];

        // Create P as neighbors of v that come later in the ordering
        vector<int> P;
        const auto& neighbors = graph.getNeighbors(v);

        for (int u : neighbors) {
            // Check if u comes after v in the ordering
            if (position[u] > i) {
                P.push_back(u);
            }
        }

            //  X  contains  neighbors  of  v  that  come  before  v  in  the
ordering
        vector<int> X;
        for (int u : neighbors) {
            if (position[u] < i) {
                X.push_back(u);
            }
        }
        // R contains just the vertex v initially
        vector<int> R = {v};

        // Run algorithm for this subproblem
        if (i < numThreads && numThreads > 1) {
            // For initial vertices, process in parallel
            enqueue([=, P = P, X = X, R = R]() mutable {
                chibaNishizeki(P, X, R);
            });
        } else {
            // Process sequentially
            chibaNishizeki(P, X, R);
        }

        if (totalMaximalCliques >= maxCliques) {
            break;
        }
    }
}
```

A.2 Bron-Kerbosch Implementation

```
void findAllMaximalCliques() {
    // Initialize
    maximalCliques.clear();
    maxCliqueSize = 0;
    cliqueDistribution.clear();
    cliqueDistribution.resize(1, 0);
    cliqueCount = 0;
    timeLimitExceeded = false;
    maxCliquesReached = false;
    startTime = chrono::high_resolution_clock::now();

    try {
          // Precompute neighborhood bitsets for small to medium-sized
graphs
        if (graph.getNumVertices() <= MAX_VERTICES) {
            precomputeNeighborhoods();
        }

        // Compute degeneracy ordering
        vector<int> degeneracyOrder = graph.degeneracyOrdering();
```

```
            // Partition vertices for parallel execution with load
balancing
         vector<vector<int>> partitions = partitionVertices(numThreads,
degeneracyOrder);

        // Create and start worker threads
        vector<thread> threads;
        for (int t = 0; t < numThreads; t++) {

threads.emplace_back(&BronKerboschAlgorithm::workerFunction, this,
                                                        ref(partitions[t]),
ref(degeneracyOrder));
        }

        // Wait for all threads to finish
        for (auto& t : threads) {
            if (t.joinable()) {
                t.join();
            }
        }
    } catch (const exception& e) {
          cerr << "Exception in clique enumeration: " << e.what() <<
endl;
    }
}
```

A.3 Tomita Implementation

```
void findAllMaximalCliques() {
    cout << "Starting Tomita algorithm..." << endl;

    maximalCliques.clear();
    maxCliqueSize = 0;
    cliqueDistribution.clear();
    cliqueDistribution.resize(1, 0);
    cliqueCount = 0;
    startTime = chrono::high_resolution_clock::now();

    try {
        // Partition vertices for parallel processing
                                vector<vector<int>>    partitions    =
partitionVertices(numThreads);

        // Create and start worker threads
        vector<thread> threads;
        for (int t = 0; t < numThreads; t++) {
                 threads.emplace_back(&TomitaAlgorithm::workerFunction,
this, partitions[t]);
        }

        // Wait for all threads to finish
        for (auto& t : threads) {
            t.join();
        }

        if (timeLimitExceeded) {
              cout << "Enumeration stopped early: Time limit exceeded"
<< endl;
                   cout  <<  "Results  are  partial  but  still  valid  for
analysis." << endl;
        } else if (maxCliquesReached) {
                cout  <<  "Enumeration  stopped  early:  Maximum  number  of
cliques reached" << endl;
                   cout  <<  "Results  are  partial  but  still  valid  for
analysis." << endl;
        } else {
            cout << "Complete enumeration finished normally." << endl;
        }
    } catch (const exception& e) {
          cerr  <<  "Exception  in  clique  enumeration:  "  <<  e.what()  <<
endl;
    }
```