Wrocław University of Science and Technology
Faculty of Information and Communication Technology

Field of study: **Algorithmic Computer Science**

ENGINEERING THESIS

Preferential coloring of random graphs

Karol Janic

Supervisor

Dominik Bojko, Ph. D.

Keywords: Barabási—Albert model, planarity, graph coloring, graph thickness

WROCŁAW 2024

ABSTRACT

The aim of this work was to develop methods for edge coloring of Barabási—Albert random graphs in such a way as to minimize the number of required colors while preserving the planarity of monochromatic subgraphs. An additional goal was to explore coloring with a given colors number that minimize the number of crossing edges in each subgraph. An option of coloring with a given number of colors while minimizing the number of crossings in each subgraph was also considered.

To achieve this, well-known algorithms for selecting the maximum planar subgraph were presented, and the planarity of Barabási–Albert graphs was examined by estimating the expected number of Kuratowski graph subdivisions in the random graph. Based on this, existing solutions were modified by incorporating a specific metric, and new approaches were proposed. All methods were tested and compared in terms of the quality of generated solutions as well as execution time. Significant improvements in the quality of the solutions were achieved without causing substantial changes to the computational complexity of the algorithms. Additionally, one application of such coloring was described.

STRESZCZENIE

Celem pracy było opracowanie metod kolorowania krawędziowego grafów losowych Barabási—Albert tak, aby minimalizując liczbę niezbędnych kolorów zachować planarność monochromatycznych podgrafów. Rozważano także opcję kolorowania zadaną liczbą kolorów przy minimalizacji liczby przecięć w każdym z podgrafów.

W tym celu przedstawiono znane algorytmy wybierania maksymalnego planarnego podgrafu oraz zbadano planarność grafów Barabási—Albert poprzez estymację oczekiwanej liczby podpodziałów grafów Kuratowskiego w grafie losowym. Na tej podstawie zmodyfikowano dotychczasowe znane rozwiązania poprzez dodanie do nich pewnej metryki oraz zaproponowano nowe podejścia. Wszystkie je przetestowano i porównaniu pod względem jakości generowanych rozwiązań jak i czasu działania. Udało się znacząco poprawić jakość otrzymywanych rozwiązań nie powodując znaczących zmian w złożonościach obliczeniowych danych algorytmów. Dodatkowo opisano jedno z zastosowań takiego kolorowania.

CONTENTS

INTRODUCTION

Random graphs play a significant role in modeling and analyzing complex systems, such as social networks, biological networks, or the World Wide Web. They serve as a mathematical model for graphs where connections between vertices are created randomly.

One of the important concepts in the theory of random graphs is the Barabási—Albert model [2], which allows for generating growing graphs that reflect the phenomenon of "preferential attachment." In these graphs, new vertices tend to connect to those that already have many connections, leading to the creation of a graph with vertices having a large number of neighbors (so-called "hubs").

This model also effectively describes the topology of electronic circuits [7], where a few components play a central role. In their case, planarity is a critical aspect since connections between components on a printed circuit board cannot cross. For complex systems, the circuit may consist of multiple layers, which need to be efficiently determined.

OBJECTIVE OF THE THESIS

The objective of this engineering thesis is to examine the Barabási—Albert model graphs in terms of their planarity and to develop algorithms for edge coloring of graphs to maintain the planarity of their monochromatic subgraphs or to minimize the number of edges causing a lack of planarity.

1. THEORETICAL INTRODUCTION

In this chapter, I will define the terms used in the subsequent parts of the thesis, describe the Barabási—Albert graph model, and discuss the problem of planarity and related theorems. Additionally, I will present the problem of preferential coloring and methods of evaluating its solutions.

Definition 1. *Let $[n]$ denote the set $\{1, 2, \ldots, n\}$.*

Definition 2. *Let H_n^k denote the n-th harmonic number of order k:*

$$H_n^{(k)} = \sum_{i=1}^{n} \left(\frac{1}{i}\right)^k . \tag{1.1}$$

Note: H_n denotes $H_n^{(1)}$.

Theorem 1. *The harmonic number $H_n^{(k)}$ for $k > 1$ is finite and converges to the Riemann zeta function $\zeta(k)$ as n approaches infinity.*

Definition 3. *Let f and g be functions $\mathbb{N} \to \mathbb{R}$, and let their argument n approach infinity. The notation $f = O(g)$ or $f =_O g$ indicates that the function f is asymptotically bounded by g:*

$$(\exists c > 0)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0) \quad |f(n)| < c \cdot |g(n)| . \tag{1.2}$$

The notation $f = \Theta(g)$ or $f =_\Theta g$ indicates that the functions f and g are asymptotically equivalent:

$$(\exists c_1, c_2 > 0)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0) \quad c_1 \cdot |f(n)| < |g(n)| < c_2 \cdot |f(n)| . \tag{1.3}$$

The notation $f(n) \sim g(n)$ indicates that the functions are asymptotically equivalent:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 1. \tag{1.4}$$

Definition 4. *A simple graph G is defined as a pair of disjoint sets (V, E) such that $E \subseteq V^{[2]}$, where $V^{[2]}$ is the set of all 2-element subsets of V. We say that vertices $v_1, v_2 \in V$ are connected by an edge from the set E if $\{v_1, v_2\} \in E$.*

Definition 5. *A directed graph G is defined as a pair of disjoint sets (V, E) such that $E \subseteq V \times V$, where $V \times V$ is the set of ordered pairs. We say that vertices $v_1, v_2 \in V$ are connected by an edge from the set E if $(v_1, v_2) \in E$ or $(v_2, v_1) \in E$.*

In the subsequent parts of the thesis, unless explicitly stated otherwise, assertions apply to both simple and directed graphs.

Definition 6. *A path of length $k+1$ in a graph $G = (V, E)$ is a sequence of pairwise distinct vertices $v_1, v_2, \ldots, v_k$, where each pair of consecutive vertices is connected by an edge from the set E.*

Definition 7. *A cycle in a graph $G = (V, E)$ is a path in which the first and last vertices are identical.*

Definition 8. *A graph $H = (V_H, E_H)$ is a subgraph of a graph $G = (V_G, E_G)$ if $V_H \subseteq V_G$ and $E_H \subseteq E_G$.*

Definition 9. *A simple graph $G = (V, E)$ is connected if there exists a path between every pair of its vertices.*

Definition 10. *A component $C = (V_C, E_C)$ of a simple graph $G = (V_G, E_G)$ is the largest (in terms of inclusion) possible connected subgraph of G.*

Definition 11. *A tree $T = (V_T, E_T)$ is defined as a simple graph that is acyclic and connected.*

Definition 12. *The degree of a vertex v in a simple graph G is defined as the number of edges incident to it. It is denoted as $deg_G(v)$.*

Definition 13. *The in-degree of a vertex v in a directed graph G is the number of edges directed toward it. It is denoted as $\deg_G^{\mathrm{IN}}(v)$.*

Definition 14. *The out-degree of a vertex v in a directed graph G is the number of edges directed away from it. It is denoted as $\deg_G^{\mathrm{OUT}}(v)$.*

Definition 15. *The neighborhood of a vertex v in a simple graph $G = (V, E)$ is defined as the set of vertices connected by an edge to v. It is denoted as $N_G(v)$:*

$$N_G(v) = \{u \in V : \{v, u\} \in E\} \ . \tag{1.5}$$

Definition 16. *The neighborhood of a vertex v in a directed graph $G = (V, E)$ is defined as the set of vertices to which edges lead from v. It is denoted as $N_G(v)$:*

$$N_G(v) = \{u \in V : (v, u) \in E\} \ . \tag{1.6}$$

1.1. THE BARABÁSI—ALBERT MODEL

Definition 17. *The Barabási—Albert (BA) model is a model describing the process of creating an undirected random graph G_m^n ($n, m \in \mathbb{N}^+$). It defines a sequence of graphs $G_m^{n_0}, G_m^{n_0+1}, \ldots, G_m^n$ ($n_0, m_0 \in \mathbb{N}^+$, $n_0 \leq n$, $m_0 \geq m$) such that $G_m^{n_0}$ is a fixed graph with m_0 edges and n_0 vertices numbered with elements from the set $[n_0]$, where each vertex has at least one incident edge. The graph G_m^{k+1} is obtained by adding to G_m^k a vertex numbered $k+1$ and connecting it to m different vertices $v_1, \ldots, v_m$, chosen independently and randomly, such that:*

$$\mathbb{P}(v_i = u) = \frac{\deg_{G_m^k}(u)}{\sum\limits_{v \in V_{G_m^k}} \deg_{G_m^k}(v)} = \frac{\deg_{G_m^k}(u)}{2\left|E_{G_m^k}\right|} \qquad (u \in [k]) \; . \tag{1.7}$$

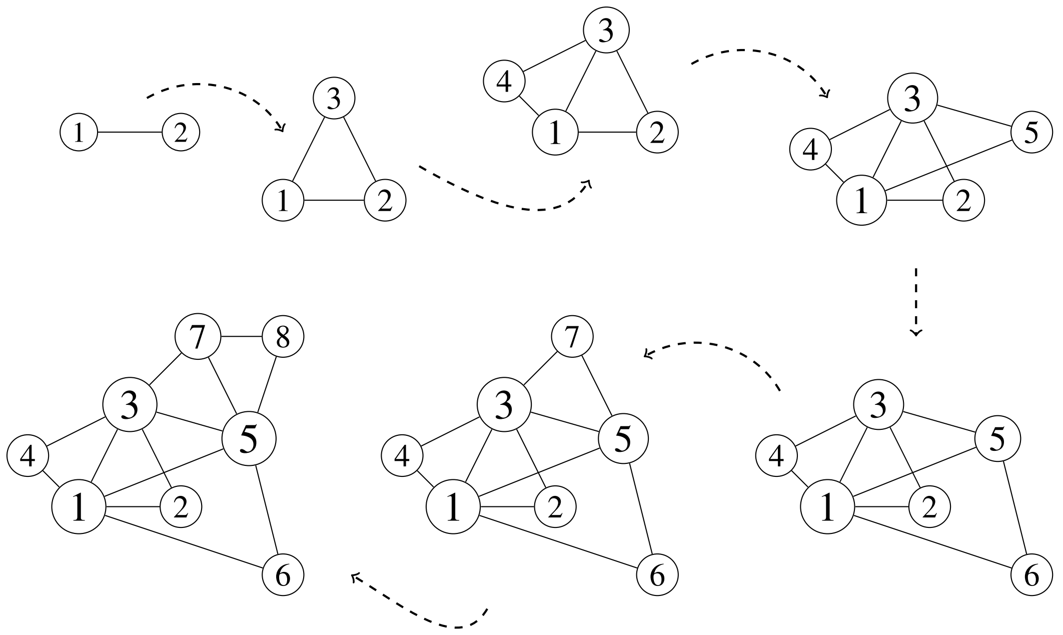


Fig. 1.1: Visualization of the process of creating the graph G_2^8 from G_2^2, which is the path P_1.

This model combines vertex growth and preferential attachment, resulting in the graph's scale-free property. This means that vertex degrees are highly varied. The degree distribution follows a power law:

$$\mathbb{P}(\deg(v) = k) \propto k^{-\gamma} \; . \tag{1.8}$$

In the case of the BA model, $\gamma = 3$. Vertices with high degrees are called hubs. The mechanism of attaching new vertices reflects Pareto's principle — a small number of vertices dominate the rest in terms of their degree, while a large number of vertices have very small degrees.

1.2. PLANARITY

This section refers to simple graphs. The theory described here can also be applied to directed graphs by converting them into simple graphs — replacing directed edges with undirected ones, removing multiple edges, and eliminating loops.

Definition 18. *A graph $G = (V, E)$ is called planar if it can be drawn on a plane such that no two edges intersect (edge intersections at vertices are allowed).*

Definition 19. *The graph K_n is a complete graph with n vertices.*

Definition 20. *The graph $K_{n,n}$ is a complete bipartite graph with $2n$ vertices.*

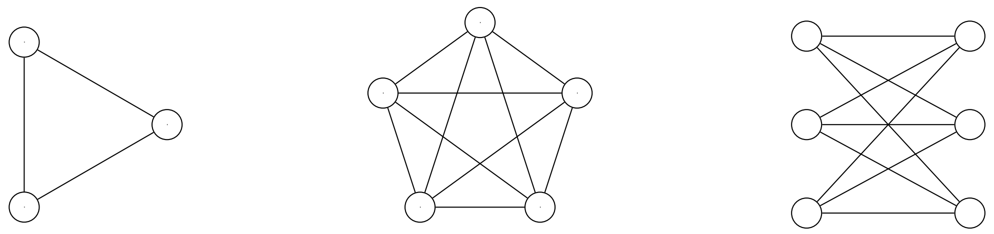


Fig. 1.2: Graphs (from left) K_3, K_5, and $K_{3,3}$.

Theorem 2. *Let $G = (V, E)$ be a simple, connected, planar graph with at least 3 vertices. Then:*

$$|E| \leq 3 \cdot |V| - 6 \; . \tag{1.9}$$

Definition 21. *An elementary subdivision of an edge $e \in E$ in a graph $G = (V, E)$ is the replacement of e with a path of length 2.*

Definition 22. *Graphs G_1 and G_2 are called homeomorphic if there exists a graph G such that both G_1 and G_2 can be obtained from G by a finite sequence of elementary subdivisions. We say that G_1 and G_2 are subdivisions of G.*

Theorem 3 (Kuratowski)**.** *A graph G is planar if and only if it does not contain a subgraph homeomorphic to $K_{3,3}$ or K_5.*

Definition 23. *The crossing number of a graph G is the smallest possible number of edge crossings in any drawing of G on a plane. It is denoted by $cr(G)$.*

Definition 24. *The thickness of a graph G is the minimum number of planar subgraphs (induced by disjoint edge sets) into which G can be divided. It is denoted by $t(G)$.*

Theorem 4. *If $G = (V, E)$ is a simple graph, then:*

$$t(G) \geq \left\lceil \frac{|E|}{3 \cdot |V| - 6} \right\rceil \; . \tag{1.10}$$

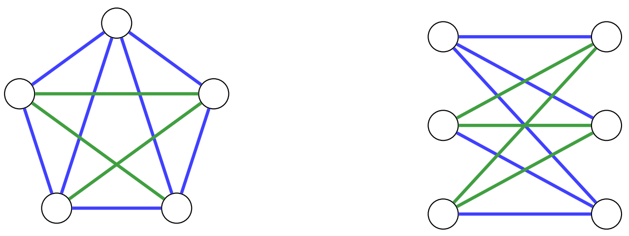


Fig. 1.3: The graphs K_5 and $K_{3,3}$ have a thickness of 2. The colors represent the division into planar subgraphs.

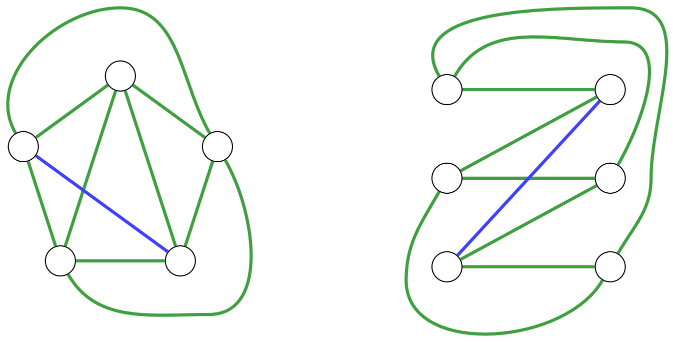


Fig. 1.4: When we try to draw the graph K_5 or $K_{3,3}$ on the plane, it turns out that we can correctly embed all edges except one. Therefore, $cr(K_5) = cr(K_{3,3}) = 1$.

The above theorem follows from Theorem 2 — no monochromatic planar subgraph can have more than $3 \cdot |V| - 6$ edges. Additionally, the thickness of a graph must be a natural number.

1.3. PREFERENTIAL COLORING OF GRAPHS

Preferential edge coloring of a graph G involves assigning each edge a color from the set $\{1, 2, \ldots, c\}$ ($c \in \mathbb{N}$) such that monochromatic subgraphs have a certain property. In our case, this property is planarity. The goal of such coloring is to use the smallest number of colors c_{MIN}.

We can observe that the minimum number of colors corresponds to the thickness of the graph G. Determining the thickness of a graph is an NP-hard problem [11]. For this reason, heuristic methods are used to obtain a solution in a reasonable time. In the literature, heuristics are based on selecting the largest planar subgraph until all edges in the original graph are exhausted. The problem of finding the maximum planar subgraph is also NP-hard [10]. Several approaches aiming to find the best solution using heuristics are present in the literature [9]. These will be presented in the next chapter.

The aforementioned minimum number of required colors is one possible measure of the quality of the coloring method. Another approach is to determine the approximation ratio of subgraphs, i.e., the ratio of the number of edges in the identified subgraph $H = (V_H, E_H)$ to the maximum number of edges in a planar graph with vertices as in the graph $G = (V_G, E_G)$ from which the planar subgraph is selected:

$$\rho_G(H) = \frac{|E_H|}{3 \cdot |V_G| - 6} \, . \tag{1.11}$$

An alternative evaluation method involves assigning a maximum number of colors, identifying monochromatic subgraphs, and determining how many edges in each must be removed to make it planar. This relates to the crossing number of a given graph G, i.e., $cr(G)$.

2. EXISTING SOLUTIONS

In this chapter, I will describe three approaches for determining maximum planar subgraphs of a given graph. These approaches are collected and described in the work [9]. Subsequently, I will describe the preferential coloring algorithm based on these approaches.

Definition 25. *A spanning tree of a graph $G = (V, E)$ is a tree that contains all vertices V, and its set of edges is a subset of the set of edges E.*

Definition 26. *A triangular cactus is a simple graph in which all cycles are triangles K_3, and every edge belongs to exactly one cycle.*

Definition 27. *A triangular structure is a graph in which all minimal cycles are triangles.*

2.1. SPANNING TREE

The simplest heuristic is to use a spanning tree. One of its properties is acyclicity. For this reason, it cannot contain a subgraph that is a subdivision of K_5 or $K_{3,3}$, as they contain at least one cycle. Using this method results in a graph H with the same vertices as G and $|V_G| - 1$ edges. Thus, $\rho_G(H) = \dfrac{|V_G| - 1}{3(|V_G| - 2)} > \dfrac{1}{3}$. A spanning tree can be found using the DFS algorithm. Its pseudocode is presented in Listing 1. The algorithm involves exploring the graph, starting from an arbitrary vertex, by traversing as far as possible until returning to an already visited vertex. In a stack-based implementation, vertices are pushed onto the stack when visiting an unvisited neighbor (lines 13-14) and popped when beginning the process of visiting them (line 11). To generate a spanning tree, edges connecting the currently visited vertex and the vertex being added to the stack must be saved (line 15). The algorithm assumes the graph is connected. In case of disconnected components, the algorithm should be applied to each component separately. Since each vertex is visited only once (ensured by the structure initialized in lines 4-7), the time complexity of the algorithm is $O(|V_G| + |E_G|)$. The memory complexity is linear with respect to the number of vertices, as each vertex is pushed onto the stack only once. A spanning tree determined using the described algorithm is marked in Figure 2.1.

2.2. THE "CACTUS" STRUCTURE

The next solution is based on triangular cacti and triangular structures. Its pseudocode is presented in Listing 2. The solution consists of two stages. In the first stage (lines 6-20),

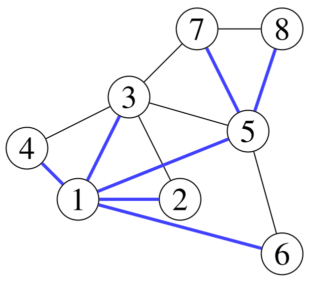


Fig. 2.1: Visualization of the spanning tree of a sample graph starting from $v_0 = 1$. Its edges are marked in blue.

Algorithm 1: SpanningTree

```
Data: connected simple graph $G = (V_G, E_G)$
Reult: spanning tree of the graph $G$
$V_T \leftarrow V_G$;
$E_T \leftarrow \emptyset$;
stack $\leftarrow \emptyset$;
visited $\leftarrow \{\}$;
foreach $v \in V_G$ do
    visited$[v]$ = NO;
end
put any vertex $v_0 \in V_G$ on the stack;
visited$[v_0]$ = YES;
// DFS simulation on the stack
while stack $\neq \emptyset$ do
    $v \leftarrow$ pop the top of the stack;
    foreach $u \in N_G(v)$ do
        if visited$[u]$ = NO then
            put $u$ on the stack;
            $E_T \leftarrow E_T \cup \{v, u\}$;
            visited$[u]$ = YES;
        end
    end
end
return $(V_T, E_T)$;
```

a maximal triangular cactus is greedily selected. Then, it is extended into a triangular structure by adding remaining edges in such a way that no new cycles are formed (lines 21-26).

The graph resulting from this algorithm is planar because it contains only cycles of length 3. Thus, it cannot contain a subgraph that is a subdivision of K_5, which has cycles of at least length 5, or a subdivision of $K_{3,3}$, whose cycles have at least length 4. The lower bound on $\rho_G(H)$ for this solution is better than the previous one, achieving $\rho_G(H) \geq \frac{7}{18}$

and can be improved to $\rho_G(H) \geq \frac{2}{5}$ by a non-greedy selection in the first stage of the algorithm [6].

An important element of this algorithm is the data structure used to track components in the resulting subgraph. It ensures that creating another triangle will not lead to other cycles. This structure should allow for creating sets, efficiently merging them, and checking if two elements are in the same set. Disjoint-set data structures serve this purpose well. Initially, each vertex belongs to a single-element component (initialization of disjoint sets is performed in lines 3-5). Then, for each vertex, triangles are selected such that all three vertices of the triangle belong to different components. This should be done iteratively to avoid generating all triangles and later checking their component membership. The three nested loops in lines 6-20 check component membership and vertex distinction "on-the-fly." After finding such a triangle, the component sets and triangular structure are updated (lines 19-20). When no further triangles can be added, edges with endpoints in different components are included, and the component sets and triangular structure are updated again (lines 25-26). This will not result in the formation of a cycle, as it would require at least two edges between components. Operations on disjoint sets have a worst-case time complexity logarithmic in the number of elements in the sets, and their amortized complexity is described by the iterated logarithm with respect to the number of vertices. This function grows very slowly, effectively making it constant in practice.

The time complexity of the algorithm consists of several factors. Initializing the component set takes $O(|V_G|)$. Scanning triangles in the worst case takes $O(|V_G|^3)$, while the final edge addition takes $O(|E_G|\log(|V_G|))$, as each component set update may take $O(\log(|V_G|))$.

The result of the described algorithm, whose pseudocode is presented in Figure 2.2, is depicted in the figure.

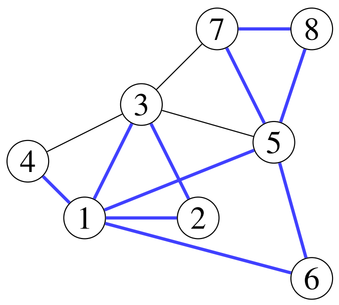


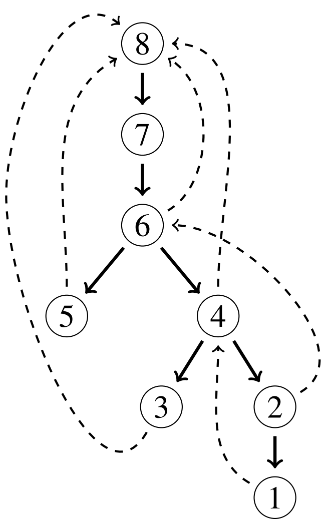
Fig. 2.2: Visualization of the cactus structure on a graph without weights. Selected triangles: 1,2,3, 1,5,6, 5,7,8 and the selected edge 1,4 are highlighted in blue.

Algorithm 2: CactusStructure

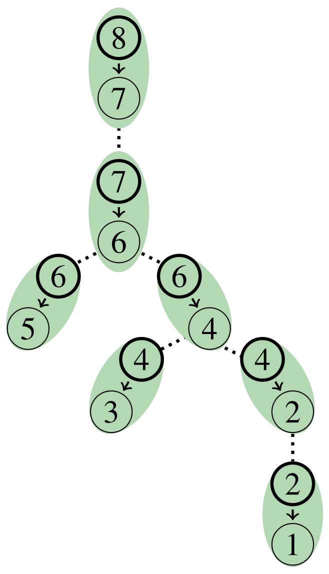
```
Data: connected simple graph $G = (V_G, E_G)$
Reult: triangular structure contained in $G$
$V_C \leftarrow \emptyset$;
$E_C \leftarrow \emptyset$;
components $\leftarrow \{\}$;   // disjoint sets structure
foreach $v \in V_G$ do
    add the set $\{v\}$ to the component;
// selecting triangles
foreach $t_1 \in V_G$ do
    $k_1 \leftarrow$ representative of the set to which $t_1$ belongs;
    foreach $t_2 \in N_G(t_1)$ do
        $k_2 \leftarrow$ representative of the set to which $t_2$ belongs;
        if $k_1 = k_2$ then
            continue;
        foreach $t_3 \in N_G(t_1)$ do
            if $t_2 = t_3$ then
                continue;
            $k_3 \leftarrow$ representative of the set to which $t_3$ belongs;
            if $k_1 = k_3$ or $k_2 = k_3$ then
                continue;
            if $\{t_2, t_3\} \in E_G$ then
                $E_C \leftarrow E_C \cup \{\{t_1, t_2\}, \{t_1, t_3\}, \{t_2, t_3\}\}$;
                union the sets containing $t_1, t_2, t_3$;
// adding edges that do not create new cycles
foreach $\{v, u\} \in E_G$ do
    $k_1 \leftarrow$ representative of the set to which $v$ belongs;
    $k_2 \leftarrow$ representative of the set to which $u$ belongs;
    if $k_1 \neq k_2$ then
        $E_C \leftarrow E_C \cup \{\{v, u\}\}$;
        union the sets containing $v$ and $u$;
return $(V_C, E_C)$
```

2.3. MODIFICATION OF THE PLANARITY TEST

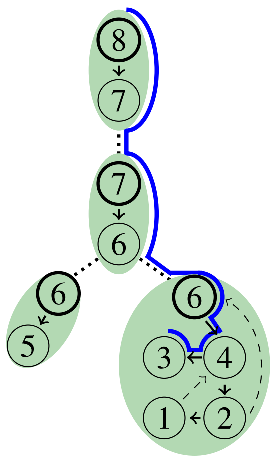
The next solution is a modification of the Boyer-Myrvold algorithm [5, 12] for testing the planarity of a graph. It is based on the construction and embedding of progressively larger blocks composed of edges from the graph $G = (V_G, E_G)$ in such a way as to maintain its planarity. The general pseudocode is presented in Listing 3, and its details are illustrated in Figures 2.3a, 2.3b, 2.3c, 2.3d, 2.3e, 2.3f.



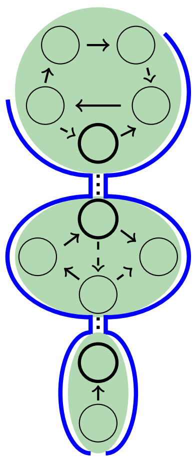
(a) Visualization of the transition tree of the DFS algorithm - the numbers represent the reverse order.



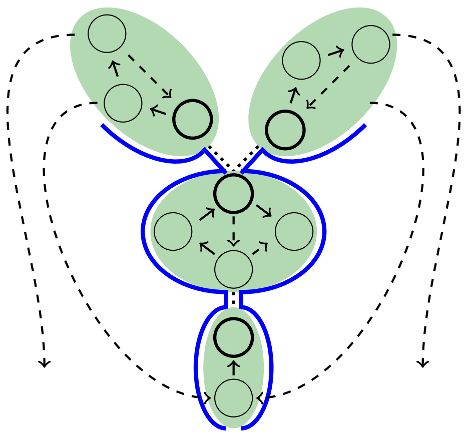
(b) Visualization of the initial blocks with selected roots (bolded vertices).



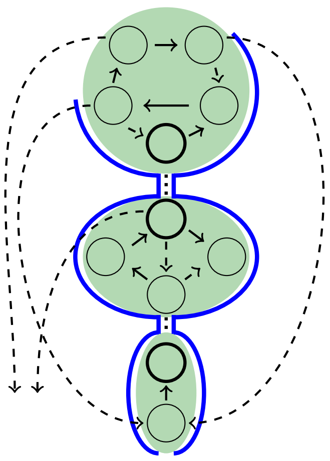
(c) Example of a potential path from vertex 8 to 3. Dashed arrows represent *backedges*.



(d) Example of two compatible paths - they do not share common edges.



(e) Example of two compatible paths - they originate from different outer blocks.



(f) Example of two incompatible paths.

The first step is to determine the reverse order of the graph's vertices during a DFS traversal and identify the backedges. This is shown in Figure Figure 2.3a, and in the pseudocode, it corresponds to lines 1-3. Next, blocks representing the edges in the DFS tree are created. For each block, the root is determined — the vertex with the highest index within it. This is visualized in Figure 2.3b. The embedding of the graph G into $\tilde{G}$ is also initialized (line 4).

Then, for each vertex v in the previously established reverse order, four phases occur: embedding blocks representing edges of the tree starting from the current vertex (lines 6-8), searching for paths (lines 9-11), embedding blocks representing the found path (lines 12-14), and checking whether the embedding of blocks and the associated backedges causes any conflicts (lines 15-19).

Path searching involves finding all paths from vertex v in the currently embedded blocks to vertices connected by backedges in the previously determined tree. The vertices forming these paths must satisfy the following properties:

1. They must lie on the boundary of the previously formed blocks,
2. The vertices that are the roots of blocks must be children of the cutting vertex in the parent block (a cutting vertex is defined as a vertex that appears in more than one block),
3. Except for the starting and ending vertices, they cannot be external vertices in the graph formed by the already selected blocks.

An example of such a path is shown in Figure 2.3c. As a result of this operation, more than one path may be selected. In this case, it is necessary to check if the selected paths are compatible, i.e., if they do not share any edges. Incompatibility can only occur when two paths pass through the same block. Suppose two paths pass through the same block. Then:

1. If the paths do not share any edges in this block, they do not share edges in any other block (Figure 2.3d),
2. If the paths pass through two different external blocks (an external block contains an external vertex in the already embedded blocks), they do not share any edge of the current block (Figure 2.3e),
3. If the paths do not share any edges of the current block and the root of the block is different from the root of the currently considered vertex, the root of the block is not an external vertex of the current embedding of blocks (Figure 2.3f).

The embedding phase involves traversing blocks from the previous iteration along the identified paths and connecting the blocks that the path contains. Additionally, backedges ending at the currently considered vertex are added to the resulting block.

If, in the phases described above, paths are found for each pair of vertices connected by a backedge, a valid embedding of the vertices is possible, and thus the graph is planar. Otherwise, the graph is not planar.

Algorithm 3: PlanarityTestBoyerMyrvold

```
   Data: simple graph $G = (V_G, E_G)$
   Reult: planarity information and embedding of the graph or
          found Kuratowski subgraph
 1 perform DFS algorithm on graph $G$;
 2 record the order of vertex visits;
 3 reverse the order of visited vertices;
 4 initialize embedding of graph $G$ - $\tilde{G}$;
 5 foreach $v$ in the determined order do
 6     foreach child $c$ of vertex $v$ in the DFS tree do
 7         add the component representing the tree edge $(v, c)$ to $\tilde{G}$
 8     end
 9     foreach $w$ connected to $v$ in the DFS tree by a back edge do
10         search for paths in $G$ between $v$ and $w$;
11     end
12     foreach child $c$ of vertex $v$ in the DFS tree do
13         embed the path between $v$ and $c$ in $\tilde{G}$;
14     end
15     foreach $w$ connected to $v$ in the DFS tree by a back edge do
16         if $(v, w) \notin \tilde{G}$ then
               // not all back edges were successfully embedded
17             return (NON-PLANAR, $\tilde{G}$);
18         end
19     end
20 end
21 return (PLANAR, $\tilde{G}$);
```

The modification of the presented planarity test involves interrupting the test when it is impossible to embed another backedge and returning the current embedding of blocks. The time complexity of this algorithm is linear in relation to the size of the graph, because the DFS algorithm has linear time complexity, the path finding process examines each edge twice, and each vertex and edge is embedded only once. An important element of the algorithm is the data structure used to join and traverse blocks. It must allow efficient traversal of block vertices in different directions (i.e., clockwise and counterclockwise) and reversing the order of elements. A data structure called *bicomp*, described in the work [12], has this advantage. This structure is based on a cyclic list, where each element contains pointers to two neighboring elements. Additionally, a pointer to the most recently visited element in the list is stored, based on which the current traversal direction can be determined. This allows for changing the "orientation" of the list in constant time. The memory complexity is also linear in relation to the size of the graph, influenced by the use of the DFS algorithm and the data structure managing the blocks.

2.4. MAXIMIZING THE SUBGRAPH

The solutions presented above select a subgraph that is planar. However, there may still be edges from the original graph whose addition does not interfere with the planarity of the subgraph. To address this, we can check for each unselected edge whether its addition preserves planarity. If so, we include it in the subgraph. The time complexity of this process is $O(|V_G|^2)$, since the planarity test can be performed in $O(|V_G| + |E_G|) = O(|V_G| + m \cdot |V_G|) = O(|V_G|)$, and the number of potentially suitable edges is $O(|E_G|) = O(m \cdot |V_G|) = O(|V_G|)$. The pseudocode for this solution is shown in Listing 4.

Algorithm 4: SubgraphMaximization

```
Data: base simple graph $G = (V_G, E_G)$
Reult: selected planar subgraph $H = (V_H, E_H)$
Reult: maximum planar subgraph $H^*$
$V_{H^*} \leftarrow V_H$;
$E_{H^*} \leftarrow E_H$;
foreach $e \in E_G \setminus E_H$ do
    $E_{H^*} \leftarrow E_{H^*} \cup \{e\}$;
    if isPlanar($H^*$) $=$ NO then
        $E_{H^*} \leftarrow E_{H^*} \setminus \{e\}$;
    end
end
return $(V_{H^*}, E_{H^*})$;
```

2.5. PREFERENTIAL COLORING

Having already established heuristics that return a planar subgraph of a given graph, we can describe the preferential coloring method. This method involves selecting a planar subgraph using one of the three heuristics described above and completing it with the maximization procedure. Then, the edges of the selected subgraph are removed from the original graph, and they are assigned a common, unique color. This procedure is repeated until no edges remain. The procedure is presented in Listing 5. Its computational complexity depends on the chosen heuristic and the number of necessary colors.

Algorithm 5: PreferentialColoring

```
Data: simple graph $G = (V_G, E_G)$
Reult: edge coloring of the graph $c : E_G \to \mathbb{N}$
$c \leftarrow \emptyset$;
$G' \leftarrow G$; // algorithm operates on a copy of the input graph
$k \leftarrow 1$; // color counter
while $E_{G'} \neq \emptyset$ do
    $H \leftarrow$ planar subgraph of $G'$ chosen by one of the heuristics;
    $H^* \leftarrow$ subgraph maximization of $H$;
    $E_{G'} \to E_{G'} \setminus E_{H^*}$;
    foreach $e \in E_{H^*}$ do
        $c[e] \leftarrow k$;
    end
    $k \leftarrow k + 1$;
end
return $c$;
```

3. EXPECTED NUMBER OF SUBDIVISIONS OF $\mathbf{K_{3,3}}$ AND $\mathbf{K_5}$ IN THE GRAPH $\mathbf{G_m^n}$

In this chapter, we determine the expected number of subdivisions of $K_{3,3}$ and K_5 in the BA graph G_m^n. Based on the work *"Mathematical results on scale-free random graphs"* [4], we calculate the expected number of subgraphs $K_{3,3}$ and K_5 in G_m^n. Subsequently, we extend this result to include the number of subdivisions of $K_{3,3}$ and K_5 contained in it.

Theorem 5 (Bollobás). *Let $G_1^n = (V_G, E_G)$ be a BA graph, and $H = (V_H, E_H)$ be a directed graph with vertices from V_G, directed edges oriented from vertices with higher indices to those with lower indices, and out-degrees no greater than 1. Then, the probability that the edges E_H appear in G_1^n is:*

$$p_H = \prod_{v \in V_H} \deg_H^{\mathrm{IN}}(v)! \prod_{(u,v)\in E_H} \frac{1}{2\sqrt{uv}} \exp\left(O\left(\sum_{v\in V_H} C_H(v)^2/v\right)\right), \tag{3.1}$$

where $C_H(v) = |\{(u,w) \in E_H : u \le v \le w\}|$.

Note: *In the precise formulation, the first product refers only to vertices that have at least one incoming edge. For simplicity, this has been replaced by all vertices. This substitution does not affect the result, as* $0! = 1$.

To apply Theorem 5 to a graph with parameter $m > 1$ and n vertices, we can model it as a graph with parameter 1 instead of m and mn vertices, then group the vertices into sets of m, treating each group as a single vertex in the resulting graph. This grouping may lead to loops in the resulting graph, which are prohibited under our definition. However, a suitable choice of edges in the graph H can prevent this.

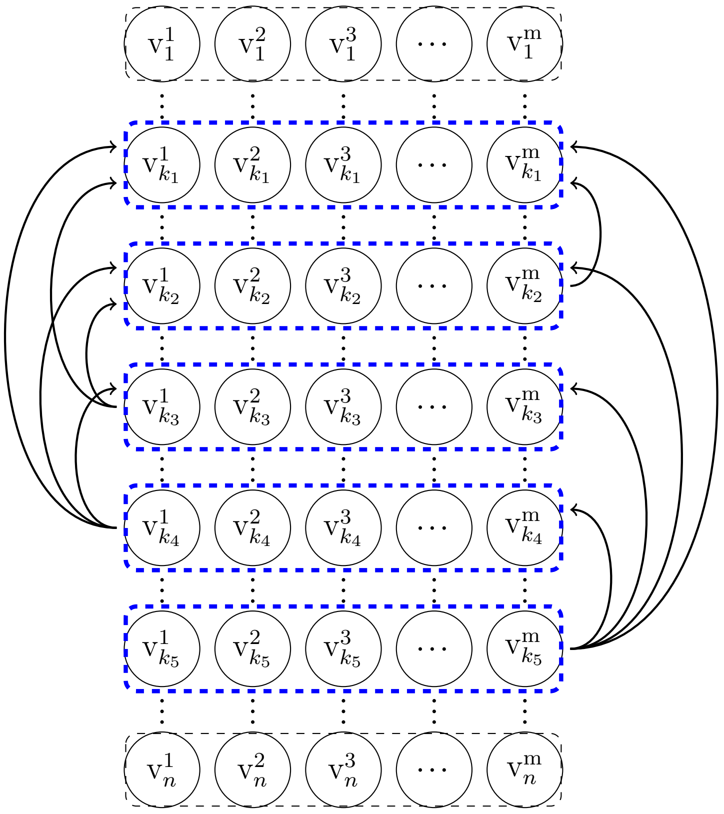
3.1. EXPECTED NUMBER OF $\mathbf{K_5}$

Consider the graph G_1^{mn} and group its vertices into n successive groups of m elements each. Define the class $\mathcal{H}$ representing directed complete graphs on 5 vertices with the following properties:

- Each vertex belongs to a different group.
- Edges are oriented to lead from vertices with higher indices to those with lower indices.
- The edges of the graph are contained in the edge set of G_1^{mn}.

- Out-degrees are no greater than 1.

Figure 3.1 illustrates the division of G_1^{mn}'s vertices into groups, the selection of groups, and the only possible arrangement of edges between them—edges connect vertices within the groups as indicated by the arrows.



v_1^1 v_1^2 v_1^3 $\cdots$ v_1^m
$v_{k_1}^1$ $v_{k_1}^2$ $v_{k_1}^3$ $\cdots$ $v_{k_1}^m$
$v_{k_2}^1$ $v_{k_2}^2$ $v_{k_2}^3$ $\cdots$ $v_{k_2}^m$
$v_{k_3}^1$ $v_{k_3}^2$ $v_{k_3}^3$ $\cdots$ $v_{k_3}^m$
$v_{k_4}^1$ $v_{k_4}^2$ $v_{k_4}^3$ $\cdots$ $v_{k_4}^m$
$v_{k_5}^1$ $v_{k_5}^2$ $v_{k_5}^3$ $\cdots$ $v_{k_5}^m$
v_n^1 v_n^2 v_n^3 $\cdots$ v_n^m

Fig. 3.1: Visualization of the vertex partition of the graph G_1^{mn} into groups and the selection of edges corresponding to the subgraph K_5 in G_m^n.

We aim to determine the expected number of elements of the class $\mathcal{H}$ and their in-degrees, which will be essential for subsequent calculations. To do this, we must find the number of ways to select vertices within the groups involved in the edges. Select one element of the class $\mathcal{H}$ and consider possible edge arrangements within the groups.

1. The arrangement of outgoing edges for vertices within group k_5 can be chosen in $m(m-1)(m-2)(m-3)$ ways.
   There are no incoming edges, so all in-degrees are 0.
2. The arrangement of outgoing edges for vertices within group k_4 can be chosen in $m(m-1)(m-2)$ ways.
   The arrangement of their incoming edges can be chosen in m ways, making one vertex have in-degree 1, and the rest 0.
3. The arrangement of outgoing edges for vertices within group k_3 can be chosen in $m(m-1)$ ways.
   The arrangement of their incoming edges can be chosen as follows:

    - $m(m-1)$ ways - two in-degrees equal to 1, the rest 0.
    - m ways - one in-degree equal to 2, the rest 0.
4. The arrangement of outgoing edges for vertices within group k_2 can be chosen in m ways.

   The arrangement of their incoming edges can be chosen as follows:
    - $m(m-1)(m-2)$ ways - three in-degrees equal to 1, the rest 0.
    - $3m(m-1)$ ways - one in-degree equal to 2, one in-degree 1, the rest 0.
    - m ways - one in-degree equal to 3, the rest 0.
5. Vertices within group k_1 have no outgoing edges.

   The arrangement of their incoming edges can be chosen as follows:
    - $m(m-1)(m-2)(m-3)$ ways - four in-degrees equal to 1, the rest 0.
    - $6m(m-1)(m-2)$ ways - one in-degree equal to 2, two in-degrees equal to 1, the rest 0.
    - $4m(m-1)$ ways - one in-degree equal to 3, one in-degree 1, the rest 0.
    - $3m(m-1)$ ways - two in-degrees equal to 2.
    - m ways - one in-degree equal to 4, the rest 0.

Let $H = (V_H, E_H)$ denote a graph from the class $\mathcal{H}$. Fix the indices of the groups $1 \le k_1 < k_2 < k_3 < k_4 < k_5 \le n$ and the numbers of vertices connected by edges: $(v_{k_5}^{l_1}, v_{k_4}^{l_1}), (v_{k_5}^{l_2}, v_{k_3}^{l_2}), (v_{k_5}^{l_3}, v_{k_2}^{l_3}), (v_{k_5}^{l_4}, v_{k_1}^{l_4}), (v_{k_4}^{l_5}, v_{k_3}^{l_5}), (v_{k_4}^{l_6}, v_{k_2}^{l_6}), (v_{k_4}^{l_7}, v_{k_1}^{l_7}), (v_{k_3}^{l_8}, v_{k_2}^{l_8}), (v_{k_3}^{l_9}, v_{k_1}^{l_9}), (v_{k_2}^{l_{10}}, v_{k_1}^{l_{10}})$— v_k^l denotes the l-th vertex of group k. We know that vertex indices within group k are bounded below by $m(k-1)$ and above by mk.

A crucial factor in computing formula 3.1 is the ratio $C_H(v)^2/v$. Since H is a fixed graph, the values of $C_H(v)$ for vertices within groups are constant. Thus:

$$\prod_{(u,v)\in E_H} 2\sqrt{uv}\exp\left(O\left(\sum_{v\in V_H} C_H(v)^2/v\right)\right) \sim 2^{10}m^{10}k_1^2k_2^2k_3^2k_4^2k_5^2 \tag{3.2}$$

Now we can determine the expected number of graphs in the class $\mathcal{H}$. For this purpose, we sum the probabilities of each of them occurring in the graph G_1^{mn}. These probabilities depend mainly on the in-degrees in the considered case. Calculating their number involves multiplying the relevant expressions describing the distribution of vertices within groups. This is a labor-intensive task, so this part of the computation was performed using a script in the Mathematica package. The script is presented in Appendix A. Additionally, various group index choices, k_1, k_2, k_3, k_4, k_5, need to be accounted for. Let $\mathcal{K}_{n,t}$ denote the set of group indices such that $1 \le k_1, k_2, \ldots, k_t \le n$, and let $\mathcal{K}_{n,t}^{\text{ord}}$ denote the set of ordered group indices such that $1 \le k_1 < k_2 < \ldots < k_t \le n$. Then:

$$
\begin{aligned}
\mathbb{E}[K_5] &\sim \sum_{\mathcal{K}_{n,5}^{\text{ord}}} \frac{(m^2-9)(m^2-4)^2(m^2-1)^3}{1024m^2(k_1k_2k_3k_4k_5)^2} \sim \\
&\sim \left(\frac{1}{5!}\right)^2 \sum_{\mathcal{K}_{n,5}} \frac{(m^2-9)(m^2-4)^2(m^2-1)^3}{1024m^2(k_1k_2k_3k_4k_5)^2} = \\
&= \frac{(m^2-9)(m^2-4)^2(m^2-1)^3}{14745600m^2}(H_n^{(2)})^5 \sim \\
&\sim \frac{(m^2-9)(m^2-4)^2(m^2-1)^3}{14745600m^2}(\zeta(2))^5
\end{aligned}
\tag{3.3}
$$

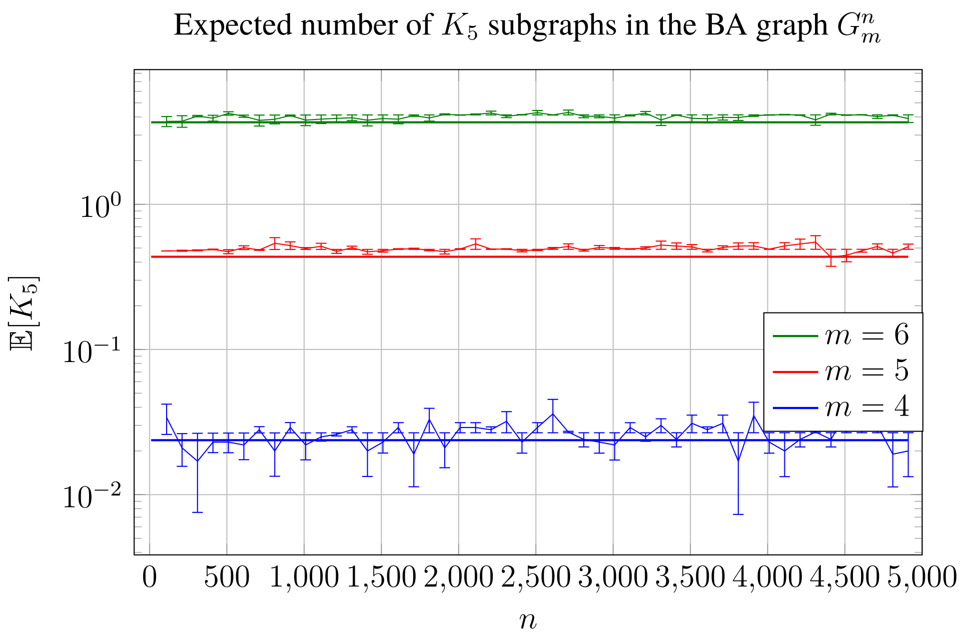


Fig. 3.2: Comparison of the expected number of K_5 subgraphs in the graph G_m^n (solid line) with the experimental results (crosses) from 100 repetitions. The vertical axis is scaled logarithmically. The vertical bars represent the standard deviation.

3.2. EXPECTED NUMBER OF $K_{3,3}$

Similarly to the reasoning presented in the previous section, calculations can be carried out to find the expected number of $K_{3,3}$ subgraphs in G_m^n. In this case, the calculations are longer because instead of one possible connection between groups with edges, there are 10. Details of the computations are presented in Appendix B.

$$\begin{aligned}\mathbb{E}[K_{3,3}] &\sim \frac{(m^2-2)^2(m^2-1)^2(5m^4-5m^2+4)}{2211840\sqrt{5}m^3}(H_n^{(3/2)})^6 \sim \\ &\sim \frac{(m^2-2)^2(m^2-1)^2(5m^4-5m^2+4)}{2211840\sqrt{5}m^3}(\zeta(3/2))^6\end{aligned} \tag{3.4}$$

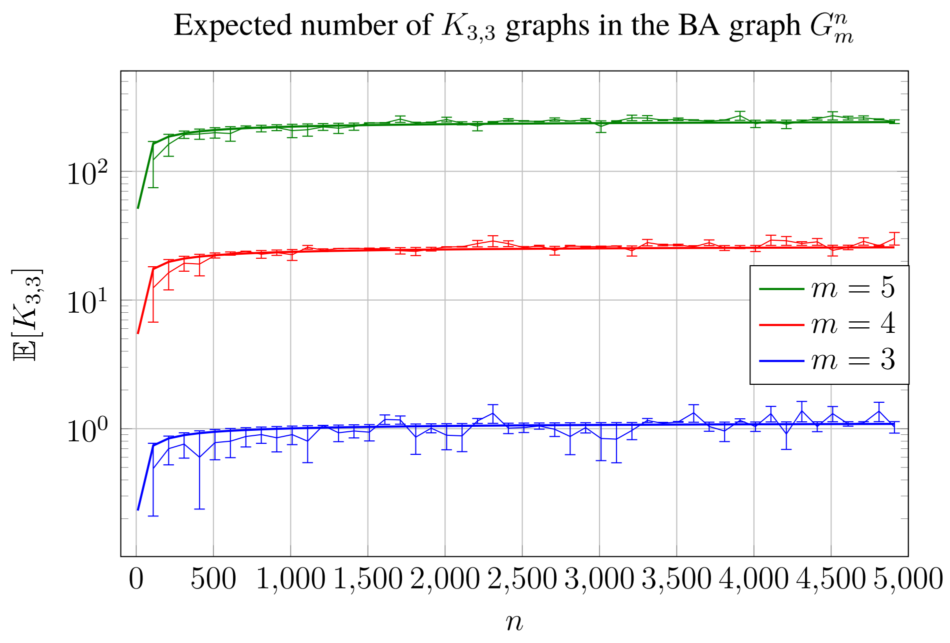


Fig. 3.3: Comparison of the expected number of $K_{3,3}$ subgraphs in the graph G_m^n (solid line) with the results obtained experimentally (crosses) in 100 repetitions. The vertical axis is logarithmically scaled. The vertical bars represent the standard deviation.

On graphs 3.2, 3.3, there is a small discrepancy between theory and experiment. To understand this, numerical simulations of individual formula components were conducted. The largest difference occurs in estimating the expected number C_H — for small numbers of vertices, it has a value closer to 1. Additionally, when summing values for each index set, the numerical value is 3 times larger than the theoretical result.

3.3. EXPECTED NUMBER OF SUBDIVISIONS OF GRAPH F

The subdivisions of a graph $F = (V_F, E_F)$ are created by adding vertices of degree 2 on its existing edges. Let $v_1, v_2, \dots, v_k$ be the vertices of the subdivision of F. Out of its vertices, we choose $|V_F|$ vertices to correspond to the vertices of the graph F — this can be done in $\binom{k}{|V_F|}$ ways. Then, the remaining vertices are divided into $|E_F|$ groups forming edges — done in $\binom{k-|V_F|+|E_F|-1}{|E_F|-1}$ ways. Each division gives a partition $k - |V_F| = k_1 + k_2 + \ldots + k_{|E_F|}$, where k_i is the number of vertices assigned to the i-th edge. The division of vertices into this partition can be realized in $\dfrac{(k-|V_F|)!}{k_1!k_2!\dots k_{|E_F|}!}$ ways. The vertices in each edge i can be arranged in $k_i!$ ways. Therefore, the number of different subdivisions of graph F with k vertices is:

$$\begin{aligned} \#F_k =& \binom{k}{|V_F|}\binom{k-|V_F|+|E_F|-1}{|E_F|-1}\frac{(k-|V_F|)!}{k_1!k_2!\dots k_{|E_F|}!}k_1!k_2!\dots k_{|E_F|}! = \\ =& \binom{k}{|V_F|}\binom{k-|V_F|+|E_F|-1}{|E_F|-1}(k-|V_F|)! \end{aligned} \tag{3.5}$$

Now, consider the graph G_1^{mn} and group its vertices into n groups of m elements each. Then, select k of them — they will form one of the $\#F_k$ subdivisions of graph F. Define edges between groups such that they correspond to the given subdivision. Let $H = (V_H, E_H)$ denote a directed graph with appropriately chosen vertices from the groups — ensuring that out-degrees do not exceed 1 and the appropriate connections between groups are maintained — and the edges corresponding to them ($V_H \subset [mn]$). The vertices in the groups forming the paths in the subdivision can then have:

1. one in-degree equal to 1 and one out-degree equal to 1 if one neighbor has a higher and the other a lower index,
2. two out-degrees equal to 1 if both neighbors have lower indices,
3. one in-degree equal to 2 or two in-degrees equal to 1 if both neighbors have higher indices.

In case 1, there are m^2 possibilities, in case 2, there are $m(m-1)$ possibilities, and in case 3, there are m or $m(m-1)$ possibilities for selecting vertices within the groups. The expected number of vertices in each of the three categories described above is equal and is given by $\dfrac{k-|V_F|}{3}$, because the indices of the vertex and its neighbors can relate in 6 different ways, and two of these fulfill each condition. Thus, the expected number of choices for vertices within the groups such that all in-degrees do not exceed 1 is:

$$(m^2)^{\frac{k-|V_F|}{3}} \cdot (m(m-1))^{\frac{k-|V_F|}{3}} \cdot (m(m-1))^{\frac{k-|V_F|}{3}} \sim m^{2(k-|V_F|)} \, . \tag{3.6}$$

The complementary case is when $\frac{k'}{3}$ vertices have an in-degree equal to 2. The expected number of such choices is:

$$(m^2)^{\frac{k-|V_F|}{3}} \cdot (m(m-1))^{\frac{k-|V_F|}{3}} \cdot m^{\frac{k-|V_F|}{3}} \sim m^{\frac{5}{3}(k-|V_F|)} \tag{3.7}$$

The degree distribution of the remaining vertices and their numbers can be carried over from the analysis of a given graph F.

To assess the probability of a specific subdivision occurring in G_1^{mn}, we also need to calculate the number $C_{F_k}(v)$ for $v \in V_H$. For each vertex v, there are v vertices with indices not greater than v and $mn - v + 1$ vertices with indices not less than v. As a result, we obtain the probability $\frac{v}{mn} \cdot \frac{mn-v+1}{mn-1}$ of the event in which one of the edges $(u, w) \in E_F$ satisfies $u \leq v \leq w$. Thus, the expected number of such edges for the entire graph F_k is:

$$\mathbb{E}[C_{F_k}(v)] = \frac{|E_{F_k}|}{mn} \sum_{v=1}^{mn} \frac{v(mn-v+1)}{(mn)(mn-1)} = |E_{F_k}| \frac{(mn+1)(mn+2)}{6(mn)(mn-1)} . \tag{3.8}$$

We can now estimate the last factor in Theorem 5:

$$\mathbb{E}\left[\sum_{v \in V_{F_k}} C_{F_k}(v)^2/v\right] \sim \left(\frac{|E_{F_k}|}{6}\right)^2 \frac{|V_{F_k}|}{mn} H_{mn} \sim \frac{|E_{F_k}|^2 |V_{F_k}|}{6} \frac{\log(mn)}{mn} \tag{3.9}$$

Fix the group indices $1 \leq l_1, l_2, \ldots, l_k \leq n$ ($\mathcal{L}_{n,k}$ denotes the set of groups) and calculate the expected number of subdivisions of graph F using k vertices in G_1^{mn}.

$$\begin{aligned}
\mathbb{E}[F_k] &=_\Theta \sum_{\mathcal{L}_{n,k'}} \#F_k \cdot \left(m^{2k'} \cdot (1!)^{\frac{k'}{3}} + m^{\frac{5}{3}k'} \cdot (2!)^{\frac{k'}{3}}\right) \cdot \frac{\exp\left(k^3 \frac{\log(mn)}{mn}\right)}{2^{k'} m^{k'} l_1 \cdot \ldots \cdot l_{k'}} \cdot \mathbb{E}[F] \\
&= \sum_{\mathcal{L}_{n,k'}} \#F_k \frac{m^{k'} + m^{\frac{2}{3}k'} \cdot 2^{\frac{k'}{3}}}{2^{k'} \cdot l_1 \cdot \ldots \cdot l_{k'}} \cdot \exp\left(k^3 \frac{\log(mn)}{mn}\right) \cdot \mathbb{E}[F] \\
&=_\Theta \#F_k \cdot \frac{m^{k'} + m^{\frac{2}{3}k'} \cdot 2^{\frac{k'}{3}}}{2^{k'}} \cdot \exp\left(k^3 \frac{\log(mn)}{mn}\right) \cdot \mathbb{E}[F] \cdot \log(n)^{k'} ,
\end{aligned} \tag{3.10}$$

where $k' = k - |V_F|$.

Note that in this case, we considered any sequence of indices, not just increasing sequences. This is due to the fact that this was already accounted for when determining the expected number of subdivisions. Finally, we can express the expected number of subdivisions H in graph F as a sum:

$$\mathbb{E}[H] = \sum_{i=|V_F|}^{n} \mathbb{E}[F_i] \tag{3.11}$$

Let us now compute values for specific graphs F.

H = subdivision of K$_5$

$$\begin{aligned}\mathbb{E}[H] &=_\Theta \sum_{k=5}^{n} \frac{k^5(k+4)!}{5!\cdot 9!} \cdot \frac{m^{k-5} + m^{\frac{2}{3}(k-5)} \cdot 2^{\frac{k-5}{3}}}{2^{k-5}} \cdot \exp\left(k^3 \frac{\log(mn)}{mn}\right) \cdot \mathbb{E}[K_5] \cdot \log(n)^k \\ &=_\Theta \sum_{k=5}^{n} \frac{k^9}{k!} \cdot \left(\left(\frac{m}{2}\right)^{k+5} + \left(\frac{m}{2}\right)^{\frac{2k+20}{3}}\right) \cdot \exp\left(k^3 \frac{\log(mn)}{mn}\right) \cdot \left(H_n^2\right)^5 \cdot \log(n)^{k-5}\end{aligned} \tag{3.12}$$

H = subdivision of K$_{3,3}$

$$\begin{aligned}\mathbb{E}[H] &=_\Theta \sum_{k=6}^{n} \frac{k^6(k+2)!}{6!\cdot 8!} \cdot \frac{m^{k-6} + m^{\frac{2}{3}k-6} \cdot 2^{\frac{k-6}{3}}}{2^{k-6}} \cdot \exp\left(k^3 \frac{\log(mn)}{mn}\right) \cdot \mathbb{E}[K_{3,3}] \cdot \log(n)^k \\ &=_\Theta \sum_{k=6}^{n} \frac{k^8}{k!} \cdot \left(\left(\frac{m}{2}\right)^{k+3} + \left(\frac{m}{2}\right)^{\frac{2k+15}{3}}\right) \cdot \exp\left(k^3 \frac{\log(mn)}{mn}\right) \cdot \left(H_n^{\frac{3}{2}}\right)^6 \cdot \log(n)^{k-6}\end{aligned} \tag{3.13}$$

3.4. OBSERVATIONS AND CONCLUSIONS

1. The expected number of K_5 graphs in the BA graph is significantly smaller than that of $K_{3,3}$ graphs.
2. As the number of vertices in the graph increases, the expected number of K_5 and $K_{3,3}$ graphs contained within it grows slowly and is upper bounded. This follows from Theorem 1.
3. The number of subdivisions of a given graph depends on the number of vertices and the parameter m of the graph G_m^n, where we count them. The tendencies of Equation 3.13 are visualized in Figure 3.4. It does not present exact values, as the result was obtained in Θ notation.
4. The distribution of the expected number of subdivisions with a given number of vertices is not uniform. It reaches high values for subdivisions with a small number of vertices. This phenomenon can be observed in Figure 3.5. It shows the experimentally determined distribution of the number of subdivisions with a given number of vertices alongside theoretical predictions. Since the theoretical formula is expressed in Θ notation, experimental and theoretical values have been normalized. The experiment used the Boyer-Myrvold graph planarity testing algorithm, which provides information about whether a graph is planar and, in the case of a negative result, identifies the subdivision causing non-planarity. This enables measuring the size of one of potentially many subdivisions in the generated graph.

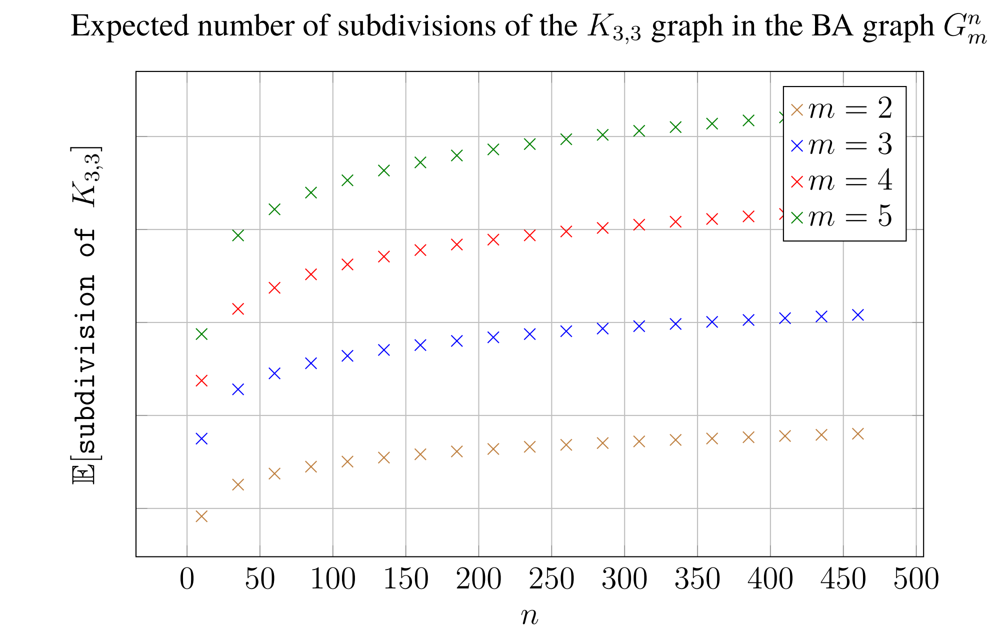


Fig. 3.4: Visualization of the theoretical values of the expected number of subdivisions of $K_{3,3}$ in the graph G_m^n. The vertical axis is logarithmically scaled.

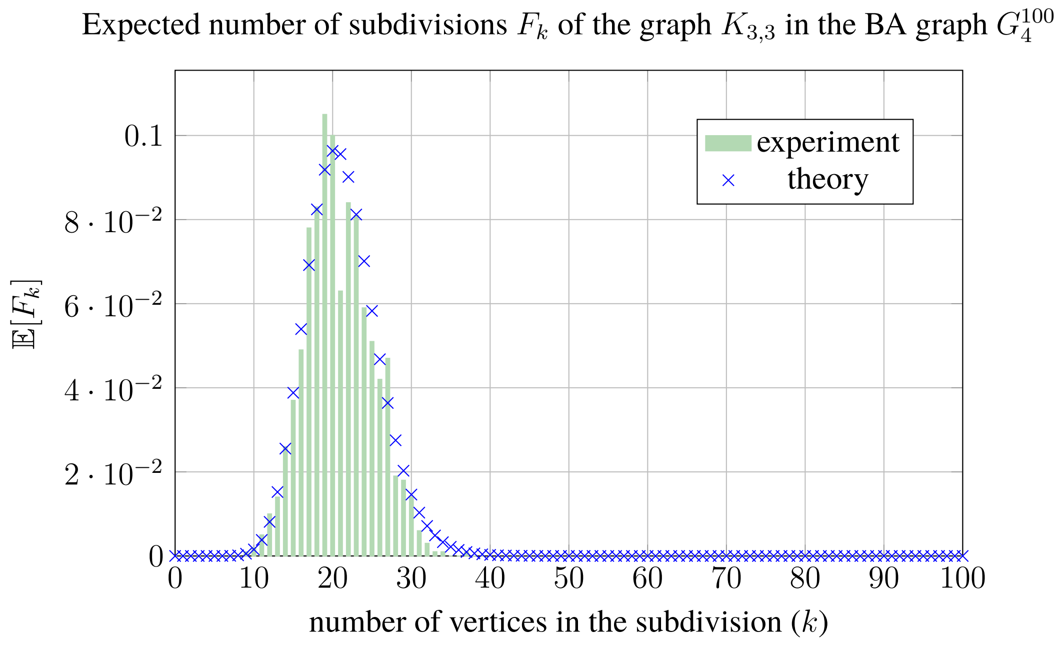


Fig. 3.5: Experimental determination of the expected number of subdivisions of the graph $K_{3,3}$ with a given number of vertices in the BA graph G_4^{100}. 1000 repetitions were performed.

3.5. STANDARD DEVIATION OF CONDUCTED EXPERIMENTS

Definition 28. *The standard deviation is a measure that determines the spread of values in the data set $x_1, \ldots, x_n$ with respect to their arithmetic mean $\overline{x}$. It is given by the following* *formula:*

$$\sigma_x = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2} \; . \tag{3.14}$$

Theorem 6 (Central Limit Theorem)**.** *Let (Ω, σ, P) be a probability space, and let $X_1, \ldots, X_n$ be independent random variables defined on Ω with the same expected value μ and the same finite standard deviation σ, and let $S_n = \sum_{i=1}^{n} X_i$. Then, as $n \to \infty$, the normalized sum S_n converges to a normal distribution (according to the distribution):*

$$Z_n \overset{d}{=} \frac{S_n - \mathbb{E}[S_n]}{\sigma_{S_n}} \overset{d}{=} \frac{S_n - n\mu}{\sigma\sqrt{n}} \overset{d}{=} \frac{\overline{X} - \mu}{\frac{\sigma}{\sqrt{n}}} \overset{d}{\to} \mathcal{N}(0,1) \; . \tag{3.15}$$

In order to better understand the expected number of K_5 and $K_{3,3}$ graphs contained in the Barabási–Albert graph, we can calculate the standard deviation. It describes how spread out the values of a random variable are around its expected value. To calculate this, we will use Theorem 6. For each number of vertices studied, a series of experiments were conducted to determine the number of these specific graphs. In each of these experiments, we can select a certain number of n distinct subsets of values, treating their means as independent random variables with the same expected value and the same finite standard deviation. Next, we compute the standard deviations within these subsets. From this, we can calculate the standard deviation of the random variable describing the number of K_5 or $K_{3,3}$ graphs in the BA graph. These values are shown in graphs 3.2 and 3.3.

4. PLANARITY INDEX

By analyzing the calculations performed in the previous chapter, we can observe that the parameters affecting the probability of a subdivision causing the non-planarity of a graph can be considered in two scales - local and global. In the local scale, the degree distribution in the graph is important - vertices with a higher number of neighbors can participate in more ways in the creation of a subdivision. In the global scale, the number of vertices n and the parameter m of the Barabási—Albert graph G_m^n are crucial.

Based on the specified local and global parameters, we can define local and global metrics, which will then be used to improve the algorithms described in the second chapter and propose a new graph coloring method.

4.1. DEFINITION

Let us take a connected graph $G_m^n = (V, E)$ and an edge $\{u, v\} \in E$. We define the following functions:

$$\tau_{G_m^n}(\{u, v\}) = \deg_{G_m^n}(u) \cdot \deg_{G_m^n}(v) \tag{4.1}$$

$$\tau(G_m^n) = \left(\sum_{\{u,v\} \in E} \tau_{G_m^n}(\{u, v\}) \right) \cdot m \cdot \log(n) \tag{4.2}$$

These defined functions take positive values. Larger values of $\tau_{G_m^n}(\{u, v\})$ mean that the edge $\{u, v\}$ has a higher probability of being used in a subgraph that is a subdivision of $K_{3,3}$ or K_5. The values of the function $\tau(G_m^n)$ for the graph indicate the likelihood of a subdivision causing non-planarity in the graph. They relate to local properties and the formulas derived in the previous chapter (3.12, 3.13). Powers have been omitted because, as observed, they do not significantly change with small changes in the number n.

4.2. APPLICATION

The metric defined for the edges of the graph can be used as a weight in the heuristics described in the second chapter, while the one defined for the entire graph can be used to evaluate different graph configurations as new edges are added. Based on this, their colors can be chosen. In the following sections, improvements to the previously described methods will be outlined, and new ones will be introduced.

4.2.1. Heuristic Improvement — Spanning Tree

The spanning tree heuristic assumed the selection of any spanning tree of a given graph G^n_m. Instead of choosing any tree, we can select a minimum spanning tree, i.e., one whose sum of edge weights is the smallest. The selected weight for the edge $\{u, v\} \in E_{G^n_m}$ will be the value $\tau_{G^n_m}(u, v)$. The tree-building process should start at the "center" of the graph — from the vertex with the highest degree. This improvement ensures that the tree consists of edges that minimize the probability of creating problematic subdivisions when later maximizing the subgraph.

Algorithm 6: MinimumSpanningTree

```
Data: connected simple graph $G = (V_G, E_G)$
Reult: spanning tree of graph $G$
$V_T \leftarrow V_G$;
$E_T \leftarrow \emptyset$;
queue $\leftarrow \emptyset$; // priority queue
visited $\leftarrow \{\}$;
foreach $v \in V_G$ do
    visited$[v]$ = NO;
end
$v_0 \leftarrow 1$;
foreach $v \in V_G$ do
    if $\deg_G(v) > \deg_G(v_0)$ then
        $v_0 \leftarrow v$;
    end
end
insert pair (NULL, $v_0$) into queue with weight 0;
while queue $\neq \emptyset$ do
    $(u, v) \leftarrow$ extract pair with minimal weight;
    if visited$[v]$ = NO then
        visited$[v]$ = YES;
        $E_T \leftarrow E_T \cup \{u, v\}$;
        foreach $w \in N_G(v)$ do
            if visited$[w]$ = NO then
                $c \leftarrow \deg_G(v) \cdot \deg_G(w)$;
                insert pair $(v, w)$ into queue with weight $c$;
            end
        end
    end
end
return $(V_T, E_T)$;
```

One of the algorithms for finding a minimum spanning tree is Prim's algorithm. The pseudocode for this solution is presented in Listing 6. It is a greedy algorithm that iteratively selects the next edges of the graph using a priority queue, until a spanning tree containing all the vertices of the graph is obtained (the condition on line 15). The first to enter the queue is the "virtual" edge. As mentioned earlier, it contains the vertex with the highest degree (its selection occurs in lines 8-13). Then, edges leading to unvisited vertices along with their corresponding weights are added (lines 20-25). The edges are removed from the queue and processed in the order of increasing weights — this ensures the operation of the priority queue (line 16). The time complexity of this algorithm depends on the implementation of the priority queue. For a binary heap, it is $O(|E_{G_m^n}| \log{(|V_{G_m^n}|)})$, since heap operations take logarithmic time relative to the number of elements in the heap, and heap updates occur for each edge in the graph. Calculating edge weights (line 22) is done in constant time. The space complexity is linear with respect to the number of vertices in the graph — the maximum number of vertices in the queue.

4.2.2. Heuristic Improvement — "Cactus" Structure

Constructing the "cactus" structure starts with selecting triangles K_3. In the previously described version, the order of their selection was arbitrary. The improvement involves selecting them in the order of increasing sums of the edge weights of the triangles. The same applies to the order of choosing edges that connect unconnected vertices with cycles. The updated algorithm using this heuristic is described in Listing 7. A significant change is generating all the triangles (lines 6 - 13), and only then adding them in sorted order (line 14), as well as sorting the graph edges before adding them (line 26). The motivation for this action is the same as in the previous case. This modification requires sorting, which increases the algorithm's time complexity by this factor. For example, the worst-case time complexity of merge sort is $O(n \log{(n)})$, where n is the number of elements to be sorted.

4.2.3. Improvement of Planar Subgraph Maximization

The subgraph maximization procedure can also be improved using the defined edge weights. These weights can be used to determine the order in which new edges are added to the subgraph. The most effective approach is to add edges in the order of decreasing weight values. The intuition behind this order is that edges added later have less impact on non-planarity. The modified algorithm is presented in Listing 8. It differs from the previous version by the addition of sorting edges in line 4. This change generates additional time complexity due to the edge sorting described in the previous subsection.

4.3. NATURAL COLORING

A coloring method that does not rely on selecting maximum planar subgraphs could be based on the structure of the Barabási—Albert graph. It involves assigning each of the m edges, which are added to the graph with each new vertex, a different color. The

Algorithm 7: WeightedCactusStructure

```
   Data: connected simple graph $G = (V_G, E_G)$
   Reult: triangular structure contained in $G$
1  $V_C \leftarrow \emptyset$;
2  $E_C \leftarrow \emptyset$;
3  components $\leftarrow \{\}$;
4  foreach $v \in V_G$ do
5      components$[v] \leftarrow v$;
6  triangles $\leftarrow \{\}$;
7  foreach $t_1 \in V_G$ do
8      foreach $t_2 \in N_G(t_1)$ do
9          foreach $t_3 \in N_G(t_1)$ do
10             if $t_2 = t_3$ then
11                 continue;
12             if $\{t_2, t_3\} \in E_G$ then
13                 triangles $\leftarrow$ triangles $\cup \{(t_1, t_2, t_3)\}$;
14 sort(triangles, ascending, key:
     $(t_1, t_2, t_3) \rightarrow \deg_G(t_1) \cdot \deg_G(t_2) + \deg_G(t_1) \cdot \deg_G(t_3) + \deg_G(t_2) \cdot \deg_G(t_3)$);
15 foreach $(t_1, t_2, t_3) \in$ triangles do
16     $k_1 \leftarrow$ components$[t_1]$;
17     $k_2 \leftarrow$ components$[t_2]$;
18     $k_3 \leftarrow$ components$[t_3]$;
19     if $k_1 \neq k_2$ and $k_1 \neq k_3$ and $k_2 \neq k_3$ then
20         $E_C \leftarrow E_C \cup \{\{t_1, t_2\}, \{t_1, t_3\}, \{t_2, t_3\}\}$;
21         $l_2 \leftarrow$ components$[t_2]$;
22         $l_3 \leftarrow$ components$[t_3]$;
23         foreach $v \in V_G$ do
24             if components$[v] = l_2 \vee$ components$[v] = l_3$ then
25                 components$[v] \leftarrow k_1$;
26 sort($E_G$, ascending, key: $(v_1, v_2) \rightarrow \deg_G(v_1) \cdot \deg_G(v_2)$);
27 foreach $\{v, u\} \in E_G$ do
28     if components$[v] \neq$ components$[u]$ then
29         $E_C \leftarrow E_C \cup \{\{v, u\}\}$;
30         $k \leftarrow$ components$[v]$;
31         $l \leftarrow$ components$[u]$;
32         foreach $w \in V_G$ do
33             if components[w] $= l$ then
34                 components[w] $= k$;
35 return $(V_C, E_C)$
```

Algorithm 8: WeightedMaximizeSubgraph

```
Data: base connected simple graph $G = (V_G, E_G)$
Reult: selected planar subgraph $H = (V_H, E_H)$
Reult: maximum planar subgraph $H^*$
V_{H^*} ← V_H;
$E_{H^*} \leftarrow E_H$;
$E \leftarrow E_G \setminus E_H$;
sort($E$, descending, key: $(v_1, v_2) \rightarrow \deg_G(v_1) \cdot \deg_G(v_2)$);
foreach $e \in E$ do
    $E_{H^*} \leftarrow E_{H^*} \cup \{e\}$;
    if isPlanar($H^*$) $=$ NO then
        $E_{H^*} \leftarrow E_{H^*} \setminus \{e\}$;
    end
end
return $(V_{H^*}, E_{H^*})$;
```

assumption that allows this operation is that the initial graph can be decomposed into no more than m planar subgraphs. Then, the described coloring will generate m trees — excluding the edges in the initial graph — and trees are planar. Colors may, but do not have to, be assigned during the graph generation process. Once the graph is generated, we can go through its vertices and assign different colors to edges leading to vertices with smaller indices. The time overhead of this method when coloring during graph generation is constant — we assign colors to m edges. In the case of coloring an already generated graph, the complexity is linear with respect to the number of edges in the graph — we visit each edge twice.

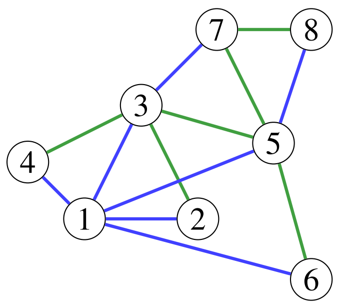


Fig. 4.1: Visualization of the natural coloring of the graph G_2^8 with two colors.

4.4. GRAPH COLORING WITH A GIVEN NUMBER OF COLORS

The approaches described above do not utilize the global parameters of the graph. We can modify the coloring task in such a way that we specify the number of colors we can use, and we want to assign edges in a way that minimizes the number of *crossing edges*.

Then, as the graph grows, we can evaluate the metric for each monochromatic subgraph induced by edges of one of the specified colors and choose the color for the new edge to minimize these metrics. Since the subgraphs may have varying densities, it is worth replacing the parameter m with the average vertex degree. An important feature of the metric τ for the graph is the ease with which we can update it. Adding a new edge requires updating the factors associated with its endpoints' neighbors as well as those related to the global properties of the graph. Due to the constant average vertex degree in the BA graph, such a metric update will, on average, take constant time.

The problem may arise from the lack of connectivity of subgraphs. In the case of a lack of connectivity, we would need to track each component separately, which could result in linear time complexity for updating the metric every time. To solve this problem, we can adopt a rule that assigns the color of an edge to the one that ensures the connectivity of one of the subgraphs in the absence of connectivity. Once the connectivity of each subgraph is ensured, the choice will be based on the metric values. In the vast majority, the color chosen based on the lack of planarity would coincide with the one chosen based on the metric. To ensure the connectivity of each subgraph, the number of edges added with each vertex must be greater than the specified number of colors. This assumption does not limit the use of the algorithm because the natural coloring described in the previous section proves that more than m colors are not needed for correct coloring. An important factor is checking the connectivity of subgraphs, which can be done using the disjoint-set data structure, as described earlier. Thus, the time complexity of this algorithm is linear in the average case with respect to the size of the initial graph.

The pseudocode for the algorithm is presented in Listing 9. It begins with the initialization of graphs, disjoint sets, and metric values for each color (lines 2-6). Then, for each edge, the metric weight is updated for each color (lines 8-16). The next phase checks whether the edge connects different components in the graph of the given color (lines 17-23) — if so, this is the desired color. Otherwise, the color corresponding to the graph with the smallest metric value is chosen (lines 24-26). Finally, the variables related to the selected color are updated (lines 27-30).

Algorithm 9: PreferentialColoringGivenColors

```
Data: BA graph $G = (V_G, E_G)$
Data: number of colors $k$
Reult: edge coloring of the graph $c : E_G \to [k]$
$c \leftarrow \emptyset$;
foreach $i \in [k]$ do
    $H_i = (V_{H_i}, E_{H_i}) \leftarrow (\emptyset, \emptyset)$; // subgraph with edges of color $i$
    $K_i \leftarrow \{\}$; // disjoint set structure of subgraph $i$
    $S_i \leftarrow 0$; // sum of $\tau$ values for edges in subgraph $i$
end
foreach $\{u, v\} \in E_G$ do
    foreach $i \in [k]$ do
        foreach $w \in N_{H_i}(u)$ do
            $S_i^* \leftarrow S_i - \deg_{H_i}(w) \cdot \deg_{H_i}(u) + \deg_{H_i}(w) \cdot (\deg_{H_i}(u) + 1)$;
        end
        foreach $w \in N_{H_i}(v)$ do
            $S_i^* \leftarrow S_i^* - \deg_{H_i}(w) \cdot \deg_{H_i}(v) + \deg_{H_i}(w) \cdot (\deg_{H_i}(v) + 1)$;
        end
        $S_i^* \leftarrow S_i^* + (\deg_{H_i}(u) + 1) \cdot (\deg_{H_i}(v) + 1)$;
    end
    $i^* \leftarrow -1$;
    foreach $i \in [k]$ do
        if $u, v$ are not in the same set in $K_i$ then
            merge the sets containing $v, u$ in $K_i$;
            $i^* \leftarrow i$;
        end
    end
    if $i^* \neq -1$ then
        $i^* \leftarrow \underset{i \in [k]}{\operatorname{argmin}} \; S_i^* \cdot \frac{|E_{H_i}|}{|V_{H_i}|} \cdot \log |V_{H_i}|$;
    end
    $V_{i^*} \leftarrow V_{i^*} \cup \{u, v\}$;
    $E_{i^*} \leftarrow E_{i^*} \cup \{\{u, v\}\}$;
    $S_{i^*} \leftarrow S_{i^*}^*$;
    $c[\{u, v\}] \leftarrow i^*$;
end
return c;
```

5. COMPARISON OF ALGORITHMS

In this chapter, we compare the previously described algorithms: selecting the largest planar subgraph, coloring a graph to preserve planar monochromatic subgraphs with the minimum number of colors, and coloring a graph with a fixed number of colors to minimize edge crossings in subgraphs. The algorithms were implemented in *C++*, with the graph structure utilizing adjacency lists. The *OGDF* library [8] was used to test graph planarity. We analyze both the quality of generated solutions and their time complexity.

A crucial part of this analysis is the method for generating Barabási—Albert graphs, which will be presented and examined in the next section.

5.1. GENERATING RANDOM BARABÁSI—ALBERT GRAPHS

The efficient method for generating a Barabási—Albert graph G_m^n with given parameters m and n is described in the work by Vladimir Batagelj and Ulrik Brandes, *Efficient generation of large random networks* [3]. It is presented here in Listing 10. The method involves creating an array of length $2mn$, where every pair of consecutive elements represents vertices connected by an edge. In this model, the probability of selecting a vertex to form a new edge is proportional to its degree, necessitating an efficient way to choose such a vertex.

For a vertex with index i ($0 \leq i < n$), the number of occurrences of index j ($0 \leq j < i$) in the array equals the current degree of vertex j in the graph formed by vertices $\{0, \ldots, i\}$ and edges described by the first $2mi$ elements of the array. Sequentially and independently, m vertices are selected for each vertex index, starting from 0 to $(n - 1)$, and these pairs are added to the generated array (lines 3-7). For the i-th vertex and the j-th edge, a pair for vertex i is chosen from the range 0-$(2mi + j)$. Finally, indices are renumbered to fall within the set $[n]$, and vertex and edge sets are created based on the arrays (lines 9-12, 13-16). This implementation requires memory proportional to the size of the graph, storing an array of size $2mn$. It also has a linear time complexity relative to the graph size, as for each of n vertices, m endpoints of new edges are randomly selected.

Figure 5.1 illustrates the experimentally observed time complexity of this algorithm, which aligns with theoretical considerations.

Algorithm 10: GenerateBAGraph

```
Data: number of vertices $n$
Data: minimum vertex degree $m$
Reult: BA graph $G = (V_G, E_G)$
M $\leftarrow$ []; // array of length 2mn
foreach $v = 0 \dots (n-1)$ do
    foreach $i = 0 \dots (m-1)$ do
        M$[2(vm+i)] \leftarrow v$;
        randomly choose $r$ from the set $\{0, \dots, 2(vm+i)\}$;
        M$[2(vm+i)] \leftarrow$ M$[r]$;
    end
end
$V_G \leftarrow \emptyset$;
foreach $i = 1 \dots n$ do
    $V_G \leftarrow V_G \cup \{i\}$;
end
$E_G \leftarrow \emptyset$;
foreach $i = 0 \dots (mn-1)$ do
    $E_G \leftarrow E_G \cup \{$M$[2i]+1,$ M$[2i+1]+1\}$;
end
return $(V_G, E_G)$;
```

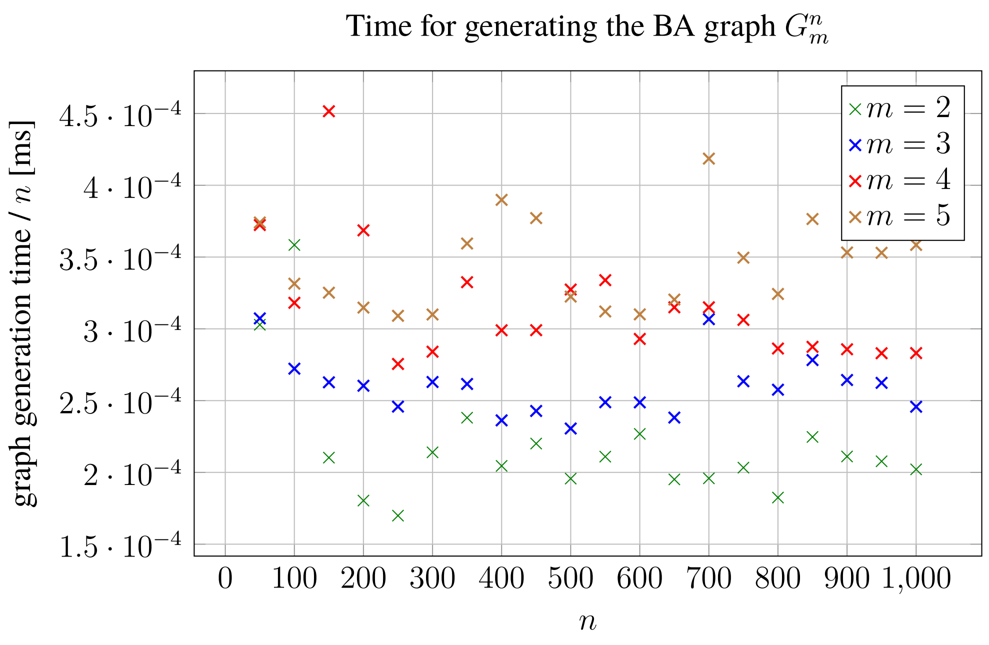


Fig. 5.1: Experimental determination of the computational complexity of the BA graph generation algorithm G^n_m. For each point, 100,000 repetitions were performed. The vertical axis represents the time in milliseconds divided by the number of vertices n.

5.2. SELECTING THE MAXIMUM PLANAR SUBGRAPH

In this section, we compare the solution quality and generation times of algorithms selecting the maximum planar subgraph, as described in Chapters 2 and 4. An experiment was conducted to generate 100 BA graphs G_m^n for various values of m and n and evaluate the size of the largest planar subgraph that could be selected. Three heuristics were applied to maximize the edge count in the planar subgraph found by the algorithm.

The results for the quality of generated solutions are presented in Figures 5.2, 5.3, 5.4, 5.5. The left axis indicates the number of edges in the subgraph (corresponding to the bars), and the right axis represents the subgraph approximation ratio (corresponding to the curves). Squares represent results from the original methods, while diamonds represent methods enhanced with the planarity index. Lighter colors depict results obtained by the original solution, while darker colors represent those enhanced by the planarity metric.

It is evident that the worst-performing algorithm is the one based on the Boyer—Myrvold planarity test, while the best is the one leveraging the "cactus" structure. Incorporating the planarity index significantly improves the spanning tree-based algorithm, increasing the number of edges in the planar subgraph by tens or hundreds, depending on the parameter m. This increase grows with the parameter value. A smaller improvement is observed for the cactus structure-based solution, where the gain grows from tens to a few dozen edges as m increases. Only a minor improvement is seen for the solution based on the planarity test. The approximation ratio reflects these edge counts but decreases with an increasing number of vertices n in every case.

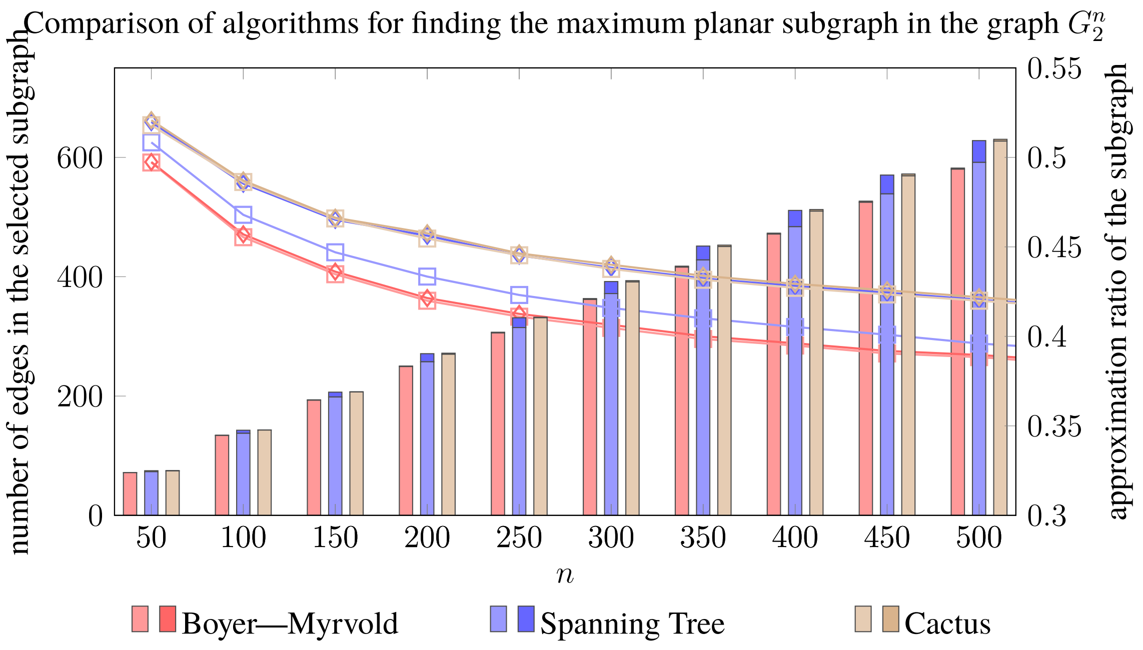


Fig. 5.2: Experimental comparison of algorithms for generating maximum planar subgraphs for the BA graph G_2^n.

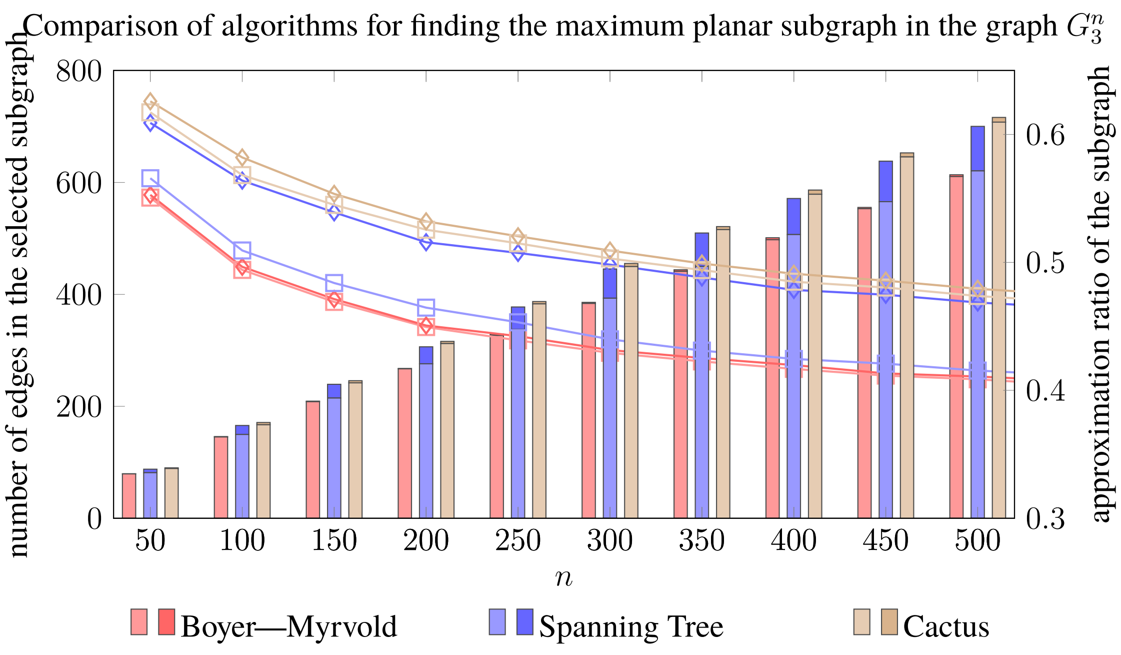


Fig. 5.3: Experimental comparison of algorithms for generating maximum planar subgraphs for the BA graph G_3^n.

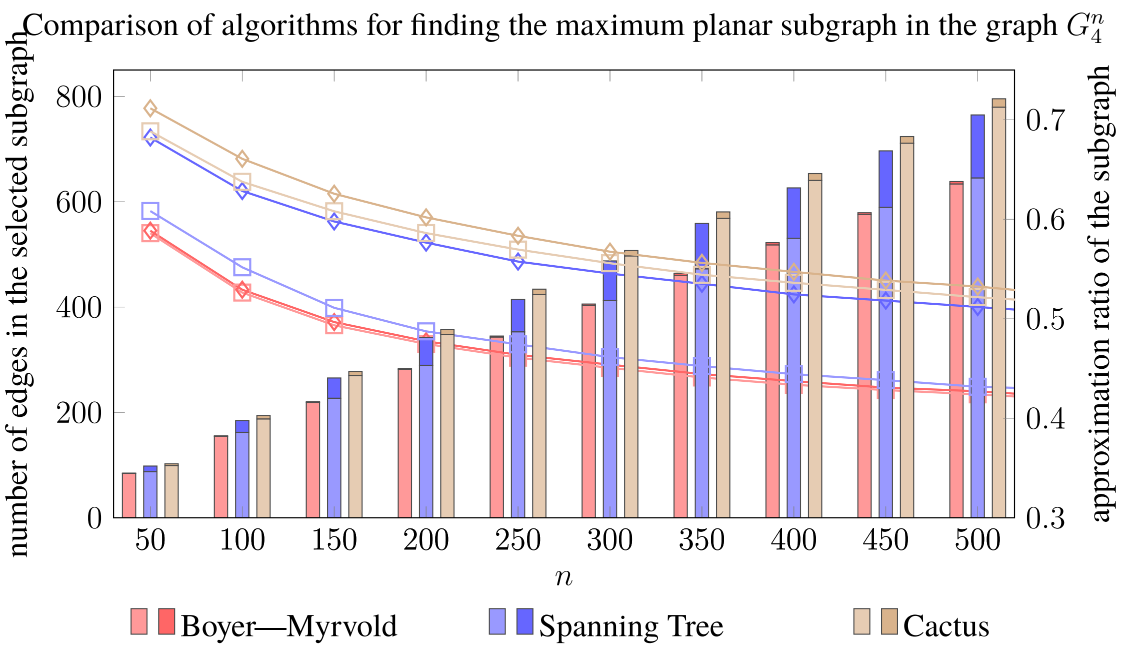


Fig. 5.4: Experimental comparison of algorithms for generating maximum planar subgraphs for the BA graph G_4^n.

Results for the runtime of the algorithms are depicted in Figures 5.6, 5.7, 5.8, and 5.9. Crosses indicate runtime for original algorithms, and pluses indicate algorithms enhanced with the planarity index. Experimental asymptotics align with theoretical predictions, with some exceptions. The Boyer—Myrvold test modification confirms linear time complexity, increasing with the parameter m, except for the value 2, possibly due to a low probability of

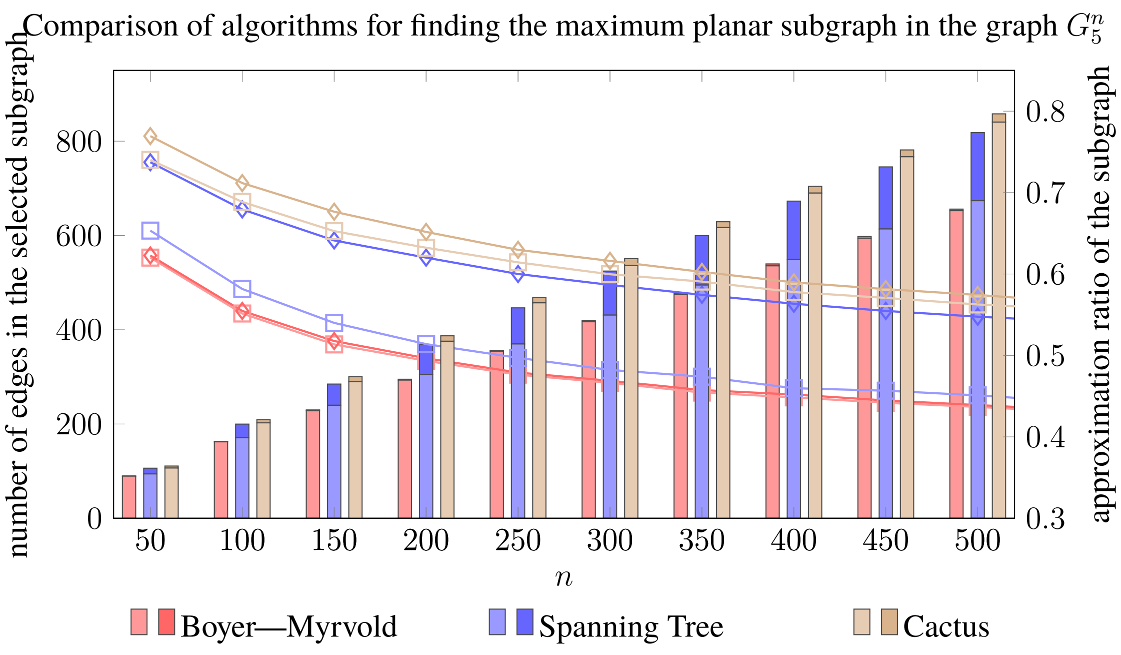


Fig. 5.5: Experimental comparison of algorithms for generating maximum planar subgraphs for the BA graph G_5^n.

K_5 or $K_{3,3}$ subdivisions in BA graphs with this parameter. Spanning-tree-based heuristics align with theoretical complexities of $O(|V_{G_m^n}|)$ and $O(|V_{G_m^n}| \log |V_{G_m^n}|)$, respectively.

For cactus-based algorithms, experimental results deviate, showing average runtime asymptotics of $|V_{G_m^n}|(\log |V_{G_m^n}|)^2$, likely due to the small number of triangles compared to the cube of vertex count and the overhead from sorting operations.

Time for selecting a planar subgraph of the BA graph G_m^n by the Boyer—Myrvold heuristic

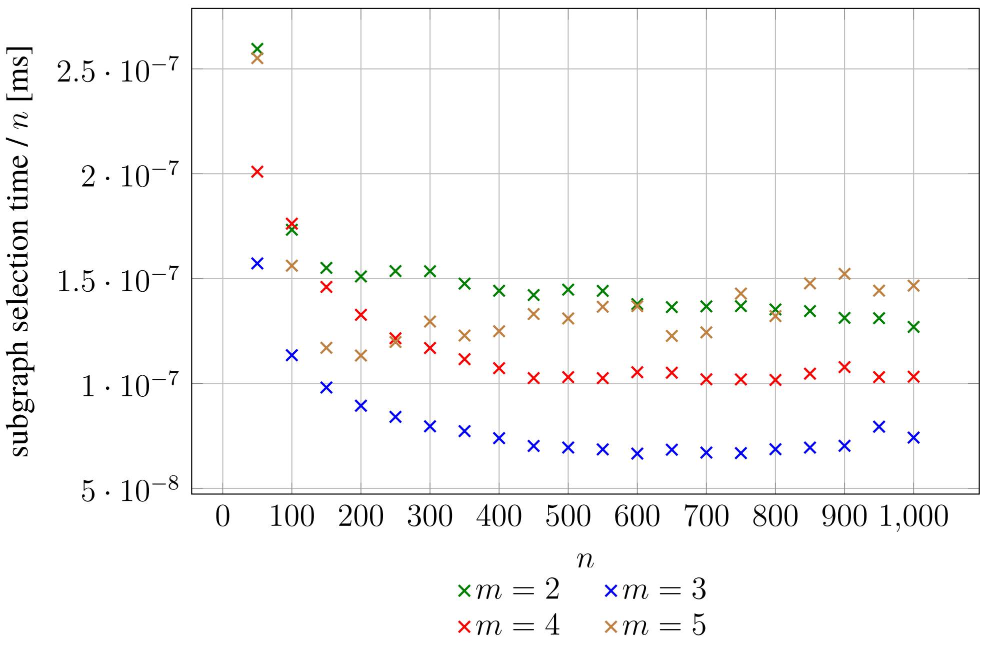


Fig. 5.6: Experimental determination of the computational complexity of the Boyer–Myrvold heuristic in the BA graph G_m^n. The vertical axis represents the time in milliseconds divided by the number of vertices n.

Time of selecting a planar subgraph of the BA graph G_m^n by the spanning tree heuristic

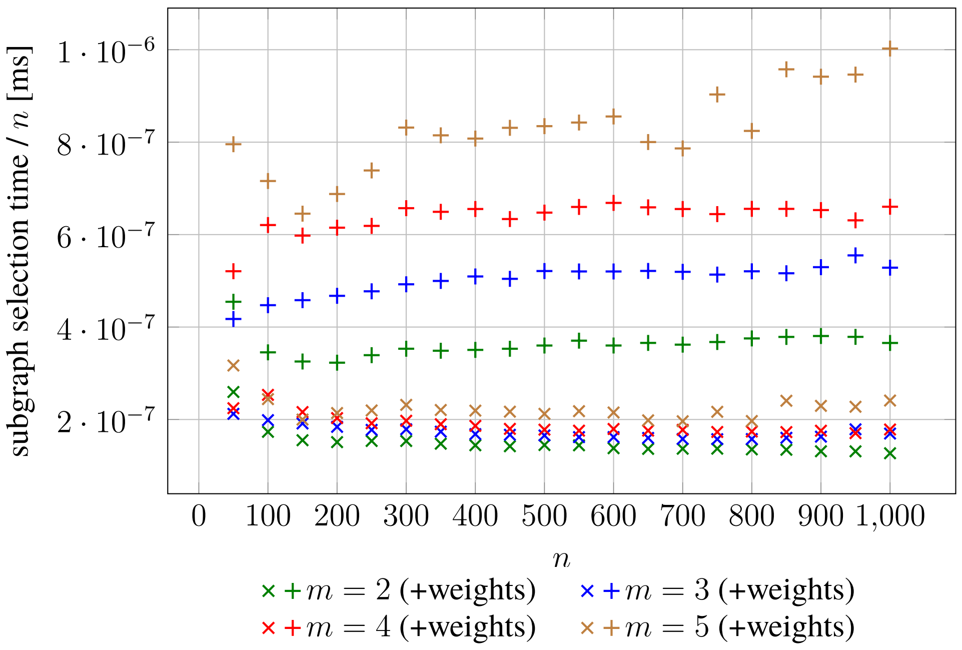


Fig. 5.7: Experimental determination of the computational complexity of the spanning tree heuristic in the BA graph G_m^n. The vertical axis represents time in milliseconds divided by the number of vertices n.

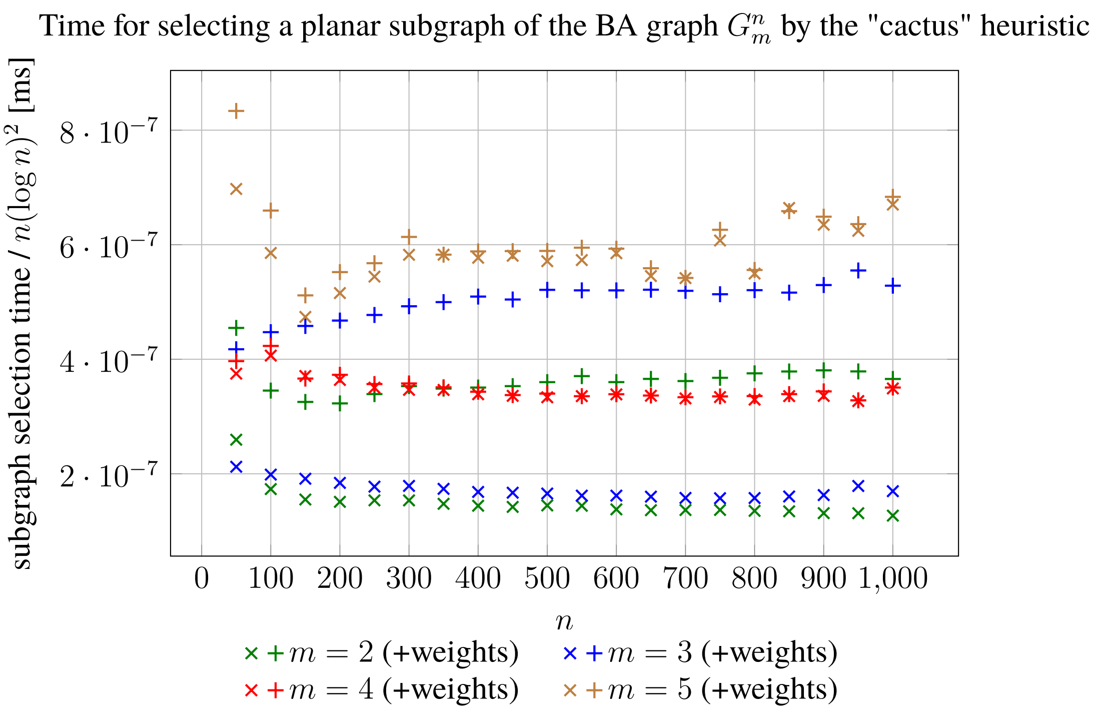


Fig. 5.8: Experimental determination of the computational complexity of the "cactus" heuristic in the BA graph G_m^n. The vertical axis represents the time in milliseconds divided by the number of vertices and the logarithm to the power of 2 — $n(\log n)^2$

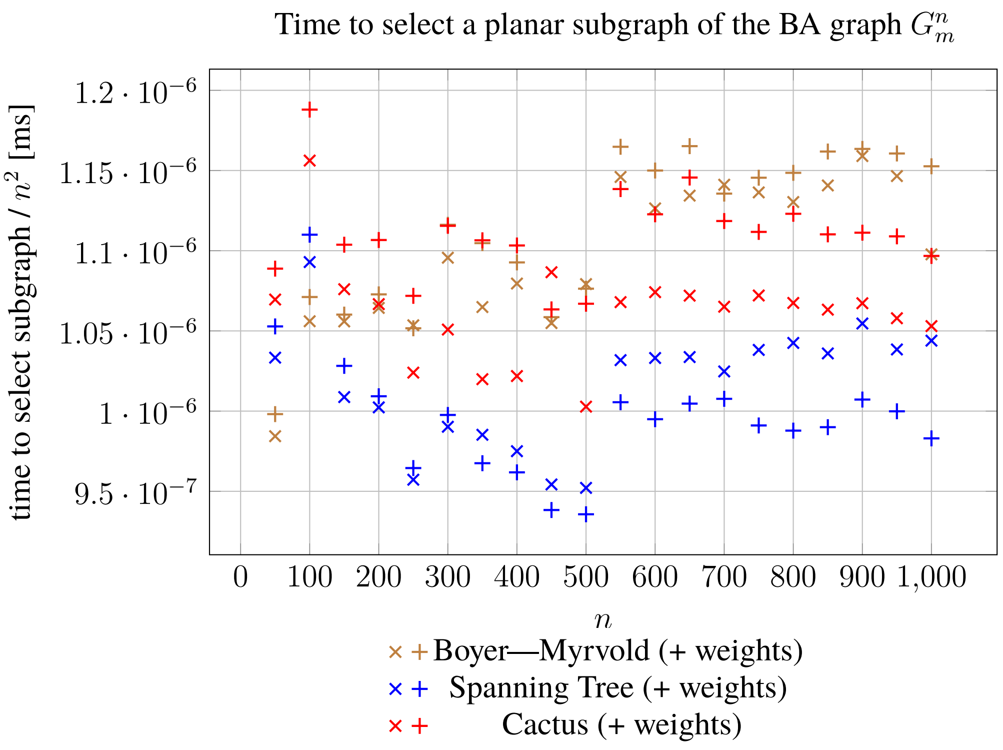


Fig. 5.9: Experimental determination of the computational complexity of the algorithm for selecting the maximum planar subgraph of the BA graph G_4^n. The vertical axis represents the time in milliseconds divided by the square of the number of vertices n^2.

5.3. COLORING WITH THE MINIMUM NUMBER OF COLORS

An important metric for evaluating the quality of algorithms is the minimum number of colors required for the preferential coloring described in the theoretical section. This also correlates with the thickness of the graph being colored. An experiment analogous to the one in the previous section was conducted, but instead of analyzing the size of the selected subgraph, the minimum number of colors was investigated. The results of this experiment are presented in Figures 5.10, 5.11, and 5.12. These figures depict two series of results for each heuristic. The series with squares represent the original method, while those with diamonds indicate the method enhanced with the planarity index.

The ranking of algorithm quality in this comparison aligns with the previous experiment. It is notable that for each value of m, the method based on "cactus" structures performs significantly better than the others. As m increases, the method relying on the minimum spanning tree demonstrates superiority over the method based on planarity testing. It is evident that for certain ranges of vertex counts in the graph G_m^n, the influence of the planarity index is significant (with the exception of the method based on planarity testing, which does not show substantial improvement). This is well illustrated in Figure 5.12, where for $n \in [100, 250]$, the improved algorithm generates solutions requiring, on average, one fewer color for correct coloring as described initially. Such ranges recur for successive values of n, and as n and m increase, the differences in the number of colors grow.

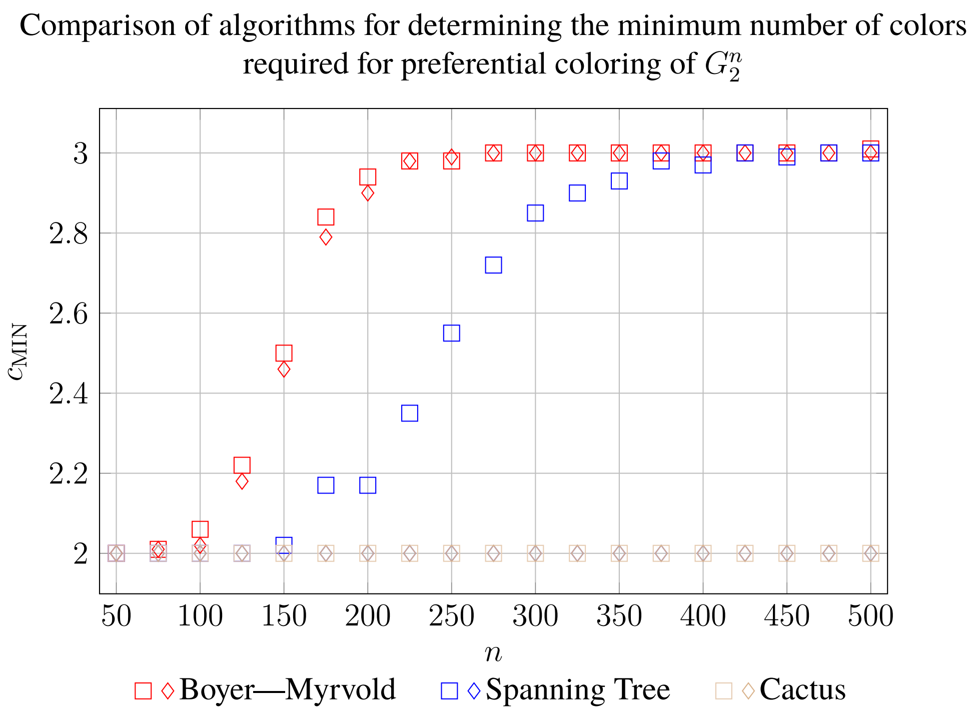


Fig. 5.10: Experimental comparison of algorithms for determining the minimum number of colors required for preferential coloring of the BA graph G_2^n.

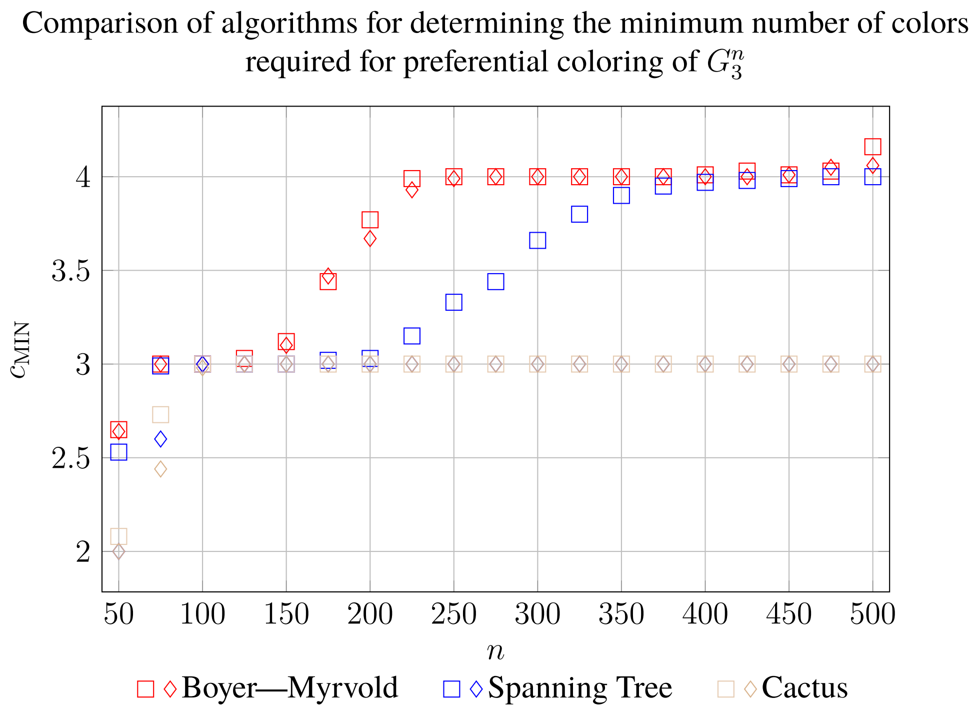


Fig. 5.11: Experimental comparison of algorithms for determining the minimum number of colors required for preferential coloring of the BA graph G_3^n.

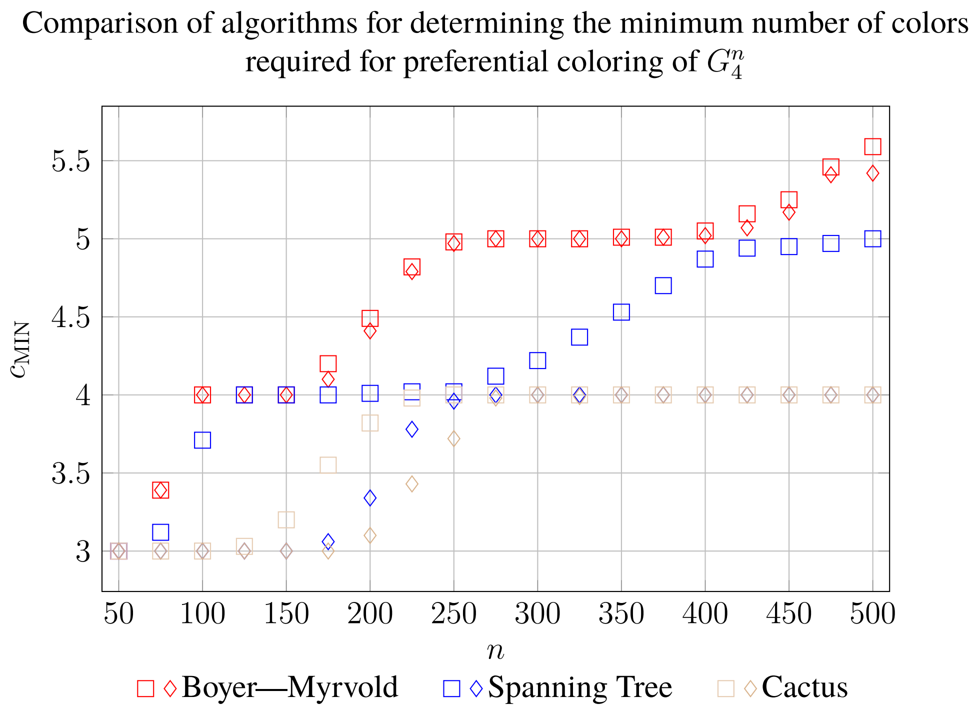


Fig. 5.12: Experimental comparison of algorithms for determining the minimum number of colors required for preferential coloring of the BA graph G_4^n.

To further track the changes in the minimum number of colors, Table 5.1 was created. It shows the ranges of vertex counts n where a given algorithm requires at least c_{MIN} colors for preferential coloring of the graph G_m^n. The symbol ∞ indicates that no upper bound for the range was established—it exceeds several tens of thousands. It is easy to observe that algorithms based on planarity testing perform the worst for the considered problem. The algorithm based on any spanning tree provides better results, but its enhancement with the planarity index delivers the highest-quality results. The best performance is achieved by the algorithm utilizing the "cactus" structure, and adding weights to it yields the best results in the entire experiment. A key observation is that for graphs with a large number of vertices, algorithms using the "cactus" heuristic and the minimum spanning tree with the presented weights require a number of colors equal to the parameter m. This result matches the outcome of the natural coloring described at the end of 2 of this work. For graphs with several tens or hundreds of vertices, depending on the value of m, it is possible to achieve coloring with fewer than m colors. In these cases, using the defined planarity index significantly improves the solution quality.

m	c_{MIN}	Boyer—Myrvold		spanning tree		"cactus" structure	
		unweighted	weighted	unweighted	weighted	unweighted	weighted
2	2	$[19, 146]$	$[19, 149]$	$[19, 240]$	$[19, \infty]$	$[19, \infty]$	$[19, \infty]$
	3	$[147, 980]$	$[150, 1000]$	$[241, \infty]$	–	–	–
3	2	$[14, 49]$	$[14, 49]$	$[14, 49]$	$[14, 49]$	$[14, 49]$	$[14, 49]$
	3	$[50, 170]$	$[50, 180]$	$[50, 270]$	$[50, \infty]$	$[50, \infty]$	$[50, \infty]$
	4	$[171, 600]$	$[181, 620]$	$[271, 1800]$	–	–	–
4	2	$[12, 27]$	$[12, 29]$	$[12, 27]$	$[12, 33]$	$[12, 32]$	$[12, 37]$
	3	$[28, 78]$	$[30, 79]$	$[28, 82]$	$[34, 227]$	$[33, 190]$	$[38, 244]$
	4	$[79, 198]$	$[80, 207]$	$[83, 338]$	$[228, \infty]$	$[190, \infty]$	$[244, \infty]$
	5	$[199, 480]$	$[208, 520]$	$[339, 1350]$	–	–	–
	6	$[481, 1250]$	$[521, 1300]$	$[1351, 2900]$	–	–	–
5	2	$[12, 21]$	$[12, 21]$	$[12, 21]$	$[12, 24]$	$[12, 22]$	$[12, 25]$
	3	$[22, 48]$	$[22, 50]$	$[22, 51]$	$[25, 79]$	$[23, 70]$	$[26, 87]$
	4	$[49, 136]$,	$[51, 138]$	$[52, 142]$	$[80, 500]$	$[71, 378]$	$[88, 588]$
	5	$[136, 231]$	$[139, 238]$,	$[143, 396]$	$[501, \infty]$	$[379, \infty]$	$[589, \infty]$
	6	$[232, 478]$	$[239, 483]$	$[397, 1200]$	–	–	–
	7	$[479, 1150]$	$[484, 1200]$	$[1201, 6300]$	–	–	–

Table 5.1: Ranges of vertex counts n in which a given algorithm requires at least c_{MIN} colors for the preferential coloring of the graph G_m^n.

5.4. COLORING WITH A GIVEN NUMBER OF COLORS

Finally, the quality of the preferential coloring algorithm with a given number of colors was examined. An experiment was conducted, generating 100 BA graphs G_m^n with different values of m and n, and a given number of colors from the range $[m]$. The runtime of the program coloring the subgraphs and the ranges of vertex counts where the average number of graph crossings was determined were analyzed. Determining these involved removing subsets of edges of sizes from 1 to n and checking whether the graph remained planar without them.

The experiment yielded the average coloring time, depicted in the graph in Figure 5.13. This confirms the theoretically established complexity of $O(|V_{G_m^n}|)$.

To compare the ranges of vertex counts, they were compiled in Table 5.2 alongside the graph parameter m, the given number of colors c, and the maximum average number of *crossing edges*. The symbol ∞ has the same meaning as in the previous table. The table shows that the developed method produces results no worse than natural coloring, which limited the number of required colors for the graph G_m^n to m. Moreover, for graphs with several tens or hundreds of vertices, it generates coloring with fewer colors on average, though it causes a few crossings in each monochromatic subgraph. The results of this method are comparable to those of the Boyer—Myrvold heuristic. Algorithms based on spanning trees and "cactus" structures produce better results. However, these three methods have higher time complexity, which may be problematic for large graphs.

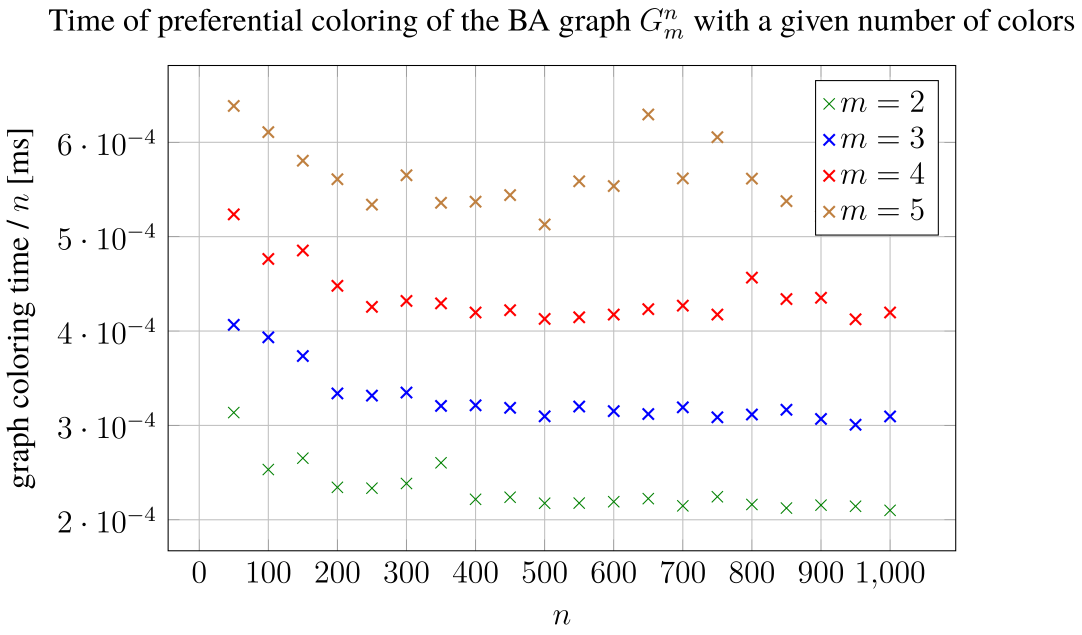


Fig. 5.13: Experimental determination of the computational complexity of the preferential coloring algorithm for the BA graph G_m^n with a given number of colors $m - 1$. Each point represents 1000 repetitions. The vertical axis shows the time in milliseconds divided by the number of vertices n.

m	c	average number of graph crossings cr					
		0	(0,1]	(1,2]	(2,3]	(3,4]	(4,5]
2	1	$[0,10]$	$[10,16]$	$[17,22]$	$[23,28]$	$[29,34]$	$[35,42]$
	≥ 2	$[0,\infty]$	$-$	$-$	$-$	$-$	$-$
3	1	$[0,7]$	$[8,11]$	$[12,13]$	$[14,15]$	$[15,16]$	$[17,17]$
	2	$[0,14]$	$[15,30]$	$[31,49]$	$[50,64]$	$[65,82]$	$[83,98]$
	≥ 3	$[0,\infty]$	$-$	$-$	$-$	$-$	$-$
4	1	$[0,6]$	$[7,8]$	$[9,9]$	$[10,10]$	$[11,11]$	$[12,12]$
	2	$[0,6]$	$[7,14]$	$[15,19]$	$[20,23]$	$[24,25]$	$[26,26]$
	3	$[0,12]$	$[13,30]$	$[31,57]$	$[58,71]$	$[72,94]$	$[95,109]$
	≥ 4	$[0,\infty]$	$-$	$-$	$-$	$-$	$-$
5	1	$[0,5]$	$[6,6]$	$[7,7]$	$[8,8]$	$[9,9]$	$[10,10]$
	2	$[0,8]$	$[9,12]$	$[13,16]$	$[17,19]$	$[20,21]$	$[22,23]$
	3	$[0,10]$,	$[11,19]$	$[20,43]$	$[44,59]$	$[60,71]$	$[72,89]$
	4	$[0,14]$	$[15,43]$,	$[44,113]$	$[114,153]$	$[154,182]$	$[182,204]$
	≥ 5	$[0,\infty]$	$-$	$-$	$-$	$-$	$-$

Table 5.2: Ranges of the number of vertices n where the preferential coloring algorithm with c colors generates at most an average of cr graph crossings for the graph G_m^n.

6. APPLICATIONS

The theory and algorithms described in previous chapters find several applications. In my opinion, the most interesting and representative of the essence of the problem is the transformation of an electronic schematic (Figure 6.1) into a printed circuit board (PCB) project (Figure 6.3). This involves converting abstract logical connections between electronic components into actual electrical connections that can be implemented on a printed circuit board.

An electronic schematic can be represented as a simple graph (Figure 6.2), where vertices represent components and edges denote connections between them. The planarity of such a graph is its most crucial property, as physical connections cannot cross—this would lead to short circuits or improper functioning of the circuit. However, the large number of vertices and edges in complex circuits typically renders such graphs non-planar. Two methods are employed to address this issue. One method is to use multilayer boards, which involve dividing the graph into the smallest possible number of planar subgraphs and then connecting them using vias (connections between different layers of the board). The other method is to use "bridges," which are additional connections that enable routing "over" an existing connection on a single layer by utilizing another layer. These solutions relate to the graph thickness and crossing number studied in this work. Layers correspond to monochromatic subgraphs generated by the described algorithms.

The family of graphs on which the proposed algorithms were tested was not arbitrary. As shown in [7], electronic schematics can effectively be modeled using Barabási—Albert random graphs. These graphs exhibit scale-free properties and "preferential attachment." This aligns with intuition since many schematics contain a few integrated circuits performing primary tasks and a large number of "auxiliary" elements (resistors, capacitors, inductors) ensuring the proper functioning of the main components.

The above arguments and the results obtained in 5 indicate that the improvements made, justified by the structure of BA graphs, have a tangible impact on the PCB design process. They can contribute to optimization in terms of both production costs (the cost of PCBs increases with each additional layer) and the electronic parameters of the circuit (lengthening the paths between components due to additional "bridges" negatively impacts signal transmission speed, energy consumption, and signal noise levels).

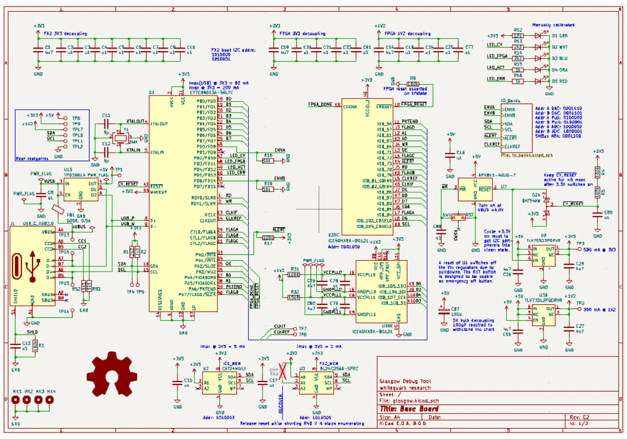


Fig. 6.1: An example electronic schematic [1] - screenshot from KiCad software.

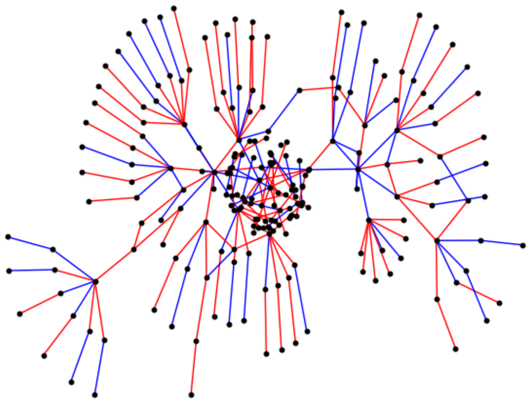


Fig. 6.2: Graph representing the above schematic with preferential coloring.

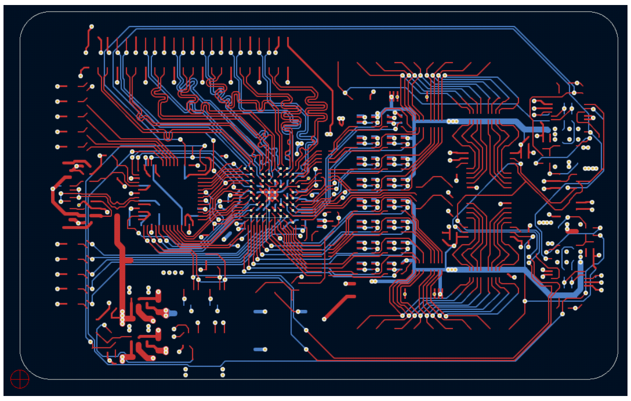


Fig. 6.3: Two-layer PCB layout [1] - screenshot from KiCad software.

SUMMARY

The goal of this thesis was to develop algorithms for edge coloring of Barabási—Albert random graphs that, while preserving the planarity of monochromatic subgraphs, minimize the number of colors or, for a given number of colors, minimize the number of intersections of each monochromatic subgraph.

Three methods currently present in the literature were presented: the modified Boyer—Myrvold planarity test, spanning trees, and the "cactus" structure. Next, the expected number of subdivisions of the graphs K_5 and $K_{3,3}$, which play key roles in Kuratowski's planarity theorem, was calculated. The results show that the expected number of K_5 and $K_{3,3}$ graphs in the BA graph is bounded above. As the number of vertices increases, their number grows in a manner similar to harmonic numbers, with orders of growth of 2 and $\frac{3}{2}$, respectively, i.e., $\frac{|E|}{|V|}$. Additionally, the distribution of the expected number of subdivisions with a given number of vertices is not uniform. It reaches high values for subdivisions where the number of vertices is a small fraction of all the vertices in the graph.

The next step was the definition of the planarity index. It was defined locally for edges and globally for the entire graph. Based on the local planarity index, the previously described algorithms were modified – it was used to determine the order in which edges should be processed. Alternative coloring methods were also described: coloring using the global definition of the index for a given number of colors, and an approach based on the Barabási—Albert graph generation process. The latter helps to limit the maximum number of colors needed for graph coloring, which in turn limits the graph's thickness: $t(G_m^n) \leq m$.

The next stage involved experimental comparison of the previously described methods and confrontation with the theory. The results show that the proposed modifications to the algorithms significantly increased the number of edges in the chosen maximal planar subgraphs, without increasing their complexity – it remained $O(|V_{G_m^n}|^2)$ before and after modification. These results translated into a reduction in the required number of colors needed for preferential coloring. The best base for maximizing the subgraph turned out to be the "cactus" heuristic, while the worst was the modification of the planarity test. The time complexity of alternative methods is lower, at $O(|V_{G_m^n}|)$, but these algorithms produce worse solutions.

Finally, the application of preferential coloring in the process of transforming an electronic schematic into a printed circuit board design was presented.

BIBLIOGRAPHY

[1] Dostępne online (6.12.2024): `https://github.com/GlasgowEmbedded/glasgow/tree/main/hardware/boards/glasgow`.

[2] Barabási, A., Albert, R., *Emergence of scaling in random networks*, tom 9781400841356 (Princeton University Press, United States, 2011), str. 349–352. Publisher Copyright: © 1999 The American Physical Society.

[3] Batagelj, V., Brandes, U., *Efficient generation of large random networks*, Physical Review E—Statistical, Nonlinear, and Soft Matter Physics. 2005, tom 71, 3, str. 036113.

[4] Bollobás, B., Riordan, O.M., *Mathematical results on scale-free random graphs*, Handbook of graphs and networks: from the genome to the internet. 2003, str. 1–34.

[5] Boyer, J.M., Myrvold, W.J., *Simplified o (n) planarity by edge addition*, Graph Algorithms Appl. 2006, tom 5, str. 241.

[6] Călinescu, G., Fernandes, C.G., Finkler, U., Karloff, H., *A better approximation algorithm for finding planar subgraphs*, Journal of Algorithms. 1998, tom 27, 2, str. 269–302.

[7] i Cancho, R.F., Janssen, C., Solé, R.V., *Topology of technology graphs: Small world patterns in electronic circuits*, Physical Review E. 2001, tom 64, 4, str. 046119.

[8] Chimani, M., Gutwenger, C., Jünger, M., Klau, G.W., Klein, K., Mutzel, P., *The open graph drawing framework (ogdf).*, Handbook of graph drawing and visualization. 2013, tom 2011, str. 543–569.

[9] Chimani, M., Klein, K., Wiedera, T., *A note on the practicality of maximal planar subgraph algorithms*, w: *Graph Drawing and Network Visualization: 24th International Symposium, GD 2016, Athens, Greece, September 19-21, 2016, Revised Selected Papers 24* (Springer, 2016), str. 357–364.

[10] Garey, M.R., Johnson, D.S., *Computers and intractability*, tom 174 (freeman San Francisco, 1979).

[11] Mansfield, A., *Determining the thickness of graphs is np-hard*, Mathematical Proceedings of the Cambridge Philosophical Society. 1983, tom 93, 1, str. 9–23.

[12] Patrignani, M., *Planarity testing and embedding*. 2013.

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF TABLES

Appendix

A. SCRIPT FOR SYMBOLICALLY CALCULATING THE EXPECTED NUMBER OF K_5 AND $K_{3,3}$ IN A GRAPH G_m^n

The appendix presents a script that was used to generate formulas defining the expected number of K_5 and $K_{3,3}$ subgraphs in a Barabási—Albert graph G_m^n. The first function is used to determine the expected number of subgraphs with fixed vertices and edges, resulting in a certain set of outgoing degrees. It generates all possible combinations of vertex distributions in the given groups. The next two listings visualize the calculation of the target values.

```
(*
cases_ - list representing vertex groups;
    its elements are lists containing pairs -
    the number of edge assignments to vertices
    and the resulting incoming degrees from such a distribution

prob_ - function that calculates the probability
    of the occurrence of a given graph in a BA graph
*)
SumUp[cases_, prob_] := Module[
    {combinations, results},

    combinations = Tuples[cases];

    results = Times @@@ (Table[
        {
            Times @@ combination[[All, 1]],
            prob[Flatten[combination[[All, 2]]]]
        },
        {combination, combinations}
    ]);

    Total[results]
]
```

```
K5SubgraphProbability[degrees_, m_] :=
    (Times @@ (Factorial /@ degrees)) / (2^10 * m^10)

K5ExpNumCoeff[m_] :=
    FullSimplify[SumUp[
        {
            { { m*(m-1)*(m-2)*(m-3), {} } },
            { { m*m*(m-1)*(m-2), {1} } },
            { { m*(m-1)*m*(m-1), {1,1} },
              { m*m*(m-1),{2} } },
            { { m*(m-1)(m-2)*m, {1,1,1} },
              { 3*m*(m-1)*m, {1,2} },
              { m*m, {3} } },
            { { m*(m-1)*(m-2)*(m-3), {1,1,1,1} },
              { 6*m*(m-1)*(m-2), {1,1,2} },
              { 4*m*(m-1), {1,3} },
              { 3*m*(m-1), {2,2} },
              { m, {4} } }
        },
        K5SubgraphProbability[#, m] &
    ]]

K5ExpectedNumber[n_, m_] :=
    K5ExpNumCoeff[m] * HarmonicNumber[n, 2]^5 / (5!)^2
```

```
K33SubgraphProbability[degrees_, m_] :=
    (Times @@ (Factorial /@ degrees)) / (2^9 * m^9)

K33ExpNumCoeff1[m_] := FullSimplify[SumUp[
    {
        { { m*(m-1)*(m-2), {} } },
        { { m*(m-1)*(m-2), {} } },
        { { m*(m-1)*(m-2), {} } },
        { { m*(m-1)*(m-2), {1,1,1} },
          { 3*m*(m-1), {1,2} },
          { m, {3} } },
        { { m*(m-1)*(m-2), {1,1,1} },
          { 3*m*(m-1), {1,2} },
          { m, {3} } },
        { { m*(m-1)*(m-2), {1,1,1} },
          { 3*m*(m-1), {1,2} },
          { m, {3} } }
    },
    K33SubgraphProbability[#, m] &
]]

K33ExpNumCoeff2[m_] := FullSimplify[SumUp[
    {
        { { m*(m-1)*(m-2), {} } },
        { { m*(m-1)*(m-2), {} } },
        { { m*m*(m-1), {1,1} },
          { m*m, {2} } },
        { { m*(m-1)*m, {1} } },
        { { m*(m-1)*(m-2), {1,1,1} },
          { 3*m*(m-1), {1,2} },
          { m, {3} } },
        { { m*(m-1)*(m-2), {1,1,1} },
          { 3*m*(m-1), {1,2} },
          { m, {3} } }
    },
    K33SubgraphProbability[#, m] &
]]

K33ExpNumCoeff3And4[m_] := FullSimplify[SumUp[
    {
        { { m*(m-1)*(m-2), {} } },
        { { m*(m-1)*m, {1} } },
        { { m*(m-1)*m, {1} } },
        { { m*(m-1)*m, {1} } },
        { { m*(m-1)*(m-2), {1,1,1} },
          { 3*m*(m-1), {1,2} },
          { m, {3} } },
        { { m*(m-1)*(m-2), {1,1,1} },
          { 3*m*(m-1), {1,2} },
          { m, {3} } }
    },
    K33SubgraphProbability[#, m] &
]]

K33ExpNumCoeff5And10[m_] := FullSimplify[SumUp[
    {
        { { m*(m-1)*(m-2), {} } },
        { { m*(m-1)*(m-2), {} } },
        { { m*m*(m-1), {} },
          { m*m, {2} } },
        { { m*m*(m-1), {1,1}},
          { m*m, {2} } },
        { { m*m*(m-1), {1,1} },
          { m*m, {2} } },
        { { m*(m-1)*(m-2), {1,1,1} },
          { 3*m*(m-1), {1,2} },
          { m, {3} } }
    },
    K33SubgraphProbability[#, m] &
]]

K33ExpNumCoeff6And7And8And9[m_] := FullSimplify[SumUp[
    {
        { { m*(m-1)*(m-2), {} } },
        { { m*(m-1)*m, {1} } },
        { { m*(m-1)*m, {1} } },
        { { m*m*(m-1), {1,1} },
          { m*m, {2} } },
        { { m*m*(m-1), {1,1} },
          { m*m, {2} } },
        { { m*(m-1)*(m-2), {1,1,1} },
          { 3*m*(m-1), {1,2} },
            { m, {3} } }
    },
    K33SubgraphProbability[#, m] &
]]

K33ExpectedNumber[n_, m_] :=
    FullSimplify[Total[
        {
            K33ExpNumCoeff1[m],
            K33ExpNumCoeff2[m],
            2 * K33ExpNumCoeff3And4[m],
            2 * K33ExpNumCoeff5And10[m],
            4 * K33ExpNumCoeff6And7And8And9[m]
        }
    ]] * HarmonicNumber[n, 3/2]^6 / (6!)^(3/2)
```

B. DETERMINING THE EXPECTED NUMBER OF $\mathrm{K}_{3,3}$ IN THE GRAPH $\mathrm{G}_\mathrm{m}^\mathrm{n}$

In this appendix, we will approximate the details of calculating the expected number of $K_{3,3}$ subgraphs in the graph G_m^n. Due to the different topology of the graph, compared to the clique considered in Chapter 3, the number of distinct connections between groups is larger.

Similarly, we start by choosing the graph G_1^{mn}, dividing its vertices into m groups, each containing n elements, and defining the class $\mathcal{H}$, which represents directed complete bipartite graphs with 6 vertices and the following properties:

- each vertex belongs to a different group,
- the edges are directed from vertices with higher indices to those with lower indices,
- the edges of the graph are contained in the edge set of the graph G_1^{mn},
- the out-degrees do not exceed 1.

Figure B.1 shows the division of vertices of G_1^{mn} into groups, the selection of groups, and one of the possible 10 edge routing scenarios between them (Scenario 1) – the edges connect certain vertices within groups, indicated by arrows. The partitions are marked in blue and green.

Now, we select an element from the class $\mathcal{H}$ and determine the number of vertex divisions within the groups and the corresponding in-degrees of those vertices. These divisions also depend on which groups we wish to connect. Let the groups be indexed as $1 \leq k_1 < k_2 < k_3 < k_4 < k_5 < k_6 \leq n$.

1. Partition selection: $A_1 = \{k_1, k_2, k_3\}$ and $B_1 = \{k_4, k_5, k_6\}$
   The distribution of the out-edges of vertices within groups k_4, k_5, k_6 can be chosen in $m(m-1)(m-2)$ ways. The distribution of in-edges for groups k_3, k_2, k_1 can be chosen as follows:
   - $m(m-1)(m-2)$ ways – 3 in-degrees equal to 1, the rest 0,
   - $3m(m-1)$ ways – 1 in-degree equal to 2, 1 in-degree equal to 1, the rest 0,
   - m ways – 1 in-degree equal to 3, the rest 0.
2. Partition selection: $A_2 = \{k_1, k_2, k_4\}$ and $B_2 = \{k_3, k_5, k_6\}$
   The distribution of the out-edges of vertices within groups k_5 and k_6 can be chosen in $m(m-1)(m-2)$ ways, in group k_3 in $m(m-1)$ ways, and in group k_4 in m ways. The distribution of in-edges for groups k_2, k_1 can be chosen as follows:

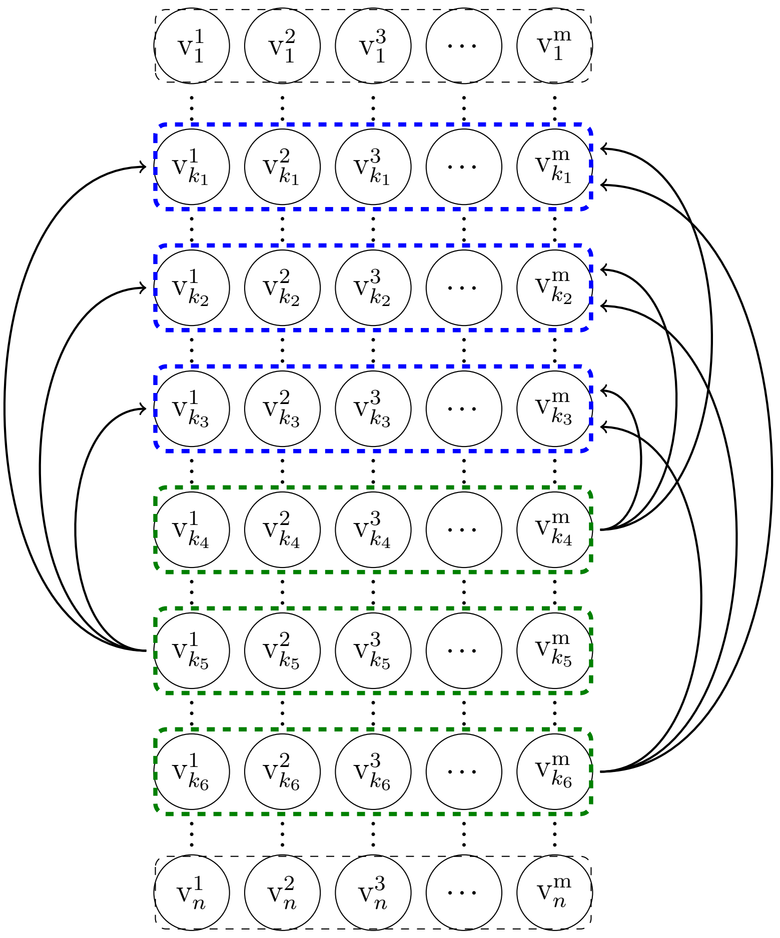


Fig. B.1: Visualization of the vertex partition of the graph G_1^{mn} into groups and one of the possible selections of edges corresponding to the subgraph $K_{3,3}$ in G_m^n.

- $m(m-1)(m-2)$ ways – 3 in-degrees equal to 1, the rest 0,
- $3m(m-1)$ ways – 1 in-degree equal to 2, 1 in-degree equal to 1, the rest 0,
- m ways – 1 in-degree equal to 3, the rest 0.

The distribution of in-edges for group k_4 can be chosen in:

- $m(m-1)$ ways – 2 in-degrees equal to 1, the rest 0,
- m ways – 1 in-degree equal to 2, the rest 0.

The distribution of in-edges for group k_3 can be chosen in m ways – one vertex with in-degree 1.

3. Partition selection: $A_3 = \{k_1, k_2, k_5\}$ and $B_3 = \{k_3, k_4, k_6\}$

   The distribution of the out-edges of vertices within group k_6 can be chosen in $m(m-1)(m-2)$ ways, within groups k_5, k_4, and k_3 in $m(m-1)$ ways.

   The distribution of in-edges for groups k_2, k_1 can be chosen as follows:

   - $m(m-1)(m-2)$ ways – 3 in-degrees equal to 1, the rest 0,
   - $3m(m-1)$ ways – 1 in-degree equal to 2, 1 in-degree equal to 1, the rest 0,
   - m ways – 1 in-degree equal to 3, the rest 0.

The distribution of in-edges for each group k_5, k_4, k_3 can be chosen in m ways – one vertex with in-degree 1.

4. Partition selection: $A_4 = \{k_1, k_2, k_6\}$ and $B_4 = \{k_3, k_4, k_5\}$

   The distribution of the out-edges of vertices within group k_6 can be chosen in $m(m-1)(m-2)$ ways, within groups k_5, k_4, and k_3 in $m(m-1)$ ways.

   The distribution of in-edges for groups k_2, k_1 can be chosen as follows:
   - $m(m-1)(m-2)$ ways – 3 in-degrees equal to 1, the rest 0,
   - $3m(m-1)$ ways – 1 in-degree equal to 2, 1 in-degree equal to 1, the rest 0,
   - m ways – 1 in-degree equal to 3, the rest 0.

   The distribution of in-edges for each group k_5, k_4, k_3 can be chosen in m ways – one vertex with in-degree 1.

5. Partition selection: $A_5 = \{k_1, k_3, k_4\}$ and $B_5 = \{k_2, k_5, k_6\}$

   The distribution of the out-edges of vertices within groups k_6, k_5 can be chosen in $m(m-1)(m-2)$ ways, within groups k_4, k_3, and k_2 in m ways.

   The distribution of in-edges for groups k_3, k_1 can be chosen as follows:
   - $m(m-1)(m-2)$ ways – 3 in-degrees equal to 1, the rest 0,
   - $3m(m-1)$ ways – 1 in-degree equal to 2, 1 in-degree equal to 1, the rest 0,
   - m ways – 1 in-degree equal to 3, the rest 0.

   The distribution of in-edges for groups k_4, k_3, k_2 can be chosen as follows:
   - $m(m-1)$ ways – 2 in-degrees equal to 1, the rest 0,
   - m ways – 1 in-degree equal to 2, the rest 0.

6. Partition choice: $A_6 = \{k_1, k_3, k_5\}$ and $B_6 = \{k_2, k_4, k_6\}$

   The distribution of the outgoing edges of the vertices within group k_6 can be chosen in $m(m-1)(m-2)$ ways, within groups k_5, k_4 in $m(m-1)$ ways, and within groups k_3, k_2 in m ways.

   The distribution of incoming edges for groups k_3, k_1 can be chosen in
   - $m(m-1)(m-2)$ ways - 3 input degrees equal to 1, the rest 0
   - $3m(m-1)$ ways - 1 input degree equal to 2, 1 input degree equal to 1, the rest 0
   - m ways - 1 input degree equal to 3, the rest 0

   The distribution of incoming edges for groups k_3, k_2 can be chosen in
   - $m(m-1)$ ways - 2 input degrees equal to 1, the rest 0
   - m ways - 1 input degree equal to 2, the rest 0

   The distribution of incoming edges in each of the groups k_5, k_4 can be chosen in m ways - with one vertex having degree 1.

7. Partition choice: $A_7 = \{k_1, k_3, k_6\}$ and $B_7 = \{k_2, k_4, k_5\}$

   The distribution of the outgoing edges of the vertices within group k_6 can be chosen in $m(m-1)(m-2)$ ways, within groups k_5, k_4 in $m(m-1)$ ways, and within groups k_3, k_2 in m ways.

   The distribution of incoming edges for group k_1 can be chosen in

- $m(m-1)(m-2)$ ways - 3 input degrees equal to 1, the rest 0
- $3m(m-1)$ ways - 1 input degree equal to 2, 1 input degree equal to 1, the rest 0
- m ways - 1 input degree equal to 3, the rest 0

The distribution of incoming edges for groups k_3, k_2 can be chosen in

- $m(m-1)$ ways - 2 input degrees equal to 1, the rest 0
- m ways - 1 input degree equal to 2, the rest 0

The distribution of incoming edges in each of the groups k_5, k_4 can be chosen in m ways - with one vertex having degree 1.

8. Partition choice: $A_8 = \{k_1, k_4, k_5\}$ and $B_8 = \{k_2, k_3, k_6\}$

   The distribution of the outgoing edges of the vertices within group k_6 can be chosen in $m(m-1)(m-2)$ ways, within groups k_5, k_4 in $m(m-1)$ ways, and within groups k_3, k_2 in m ways.

   The distribution of incoming edges for group k_1 can be chosen in

   - $m(m-1)(m-2)$ ways - 3 input degrees equal to 1, the rest 0
   - $3m(m-1)$ ways - 1 input degree equal to 2, 1 input degree equal to 1, the rest 0
   - m ways - 1 input degree equal to 3, the rest 0

   The distribution of incoming edges for groups k_3, k_2 can be chosen in

   - $m(m-1)$ ways - 2 input degrees equal to 1, the rest 0
   - m ways - 1 input degree equal to 2, the rest 0

   The distribution of incoming edges in each of the groups k_5, k_4 can be chosen in m ways - with one vertex having degree 1.

9. Partition choice: $A_9 = \{k_1, k_4, k_6\}$ and $B_9 = \{k_2, k_3, k_5\}$

   The distribution of the outgoing edges of the vertices within group k_6 can be chosen in $m(m-1)(m-2)$ ways, within groups k_5, k_4 in $m(m-1)$ ways, and within groups k_3, k_2 in m ways.

   The distribution of incoming edges for group k_1 can be chosen in

   - $m(m-1)(m-2)$ ways - 3 input degrees equal to 1, the rest 0
   - $3m(m-1)$ ways - 1 input degree equal to 2, 1 input degree equal to 1, the rest 0
   - m ways - 1 input degree equal to 3, the rest 0

   The distribution of incoming edges for groups k_3, k_2 can be chosen in

   - $m(m-1)$ ways - 2 input degrees equal to 1, the rest 0
   - m ways - 1 input degree equal to 2, the rest 0

   The distribution of incoming edges in each of the groups k_5, k_4 can be chosen in m ways - with one vertex having degree 1.

10. Partition choice: $A_{10} = \{k_1, k_5, k_6\}$ and $B_{10} = \{k_2, k_3, k_4\}$

    The distribution of the outgoing edges of the vertices within groups k_6, k_5 can be chosen in $m(m-1)(m-2)$ ways, within groups k_4, k_3, k_2 in m ways. The distribution of incoming edges for group k_1 can be chosen in

    - $m(m-1)(m-2)$ ways - 3 input degrees equal to 1, the rest 0

- $3m(m-1)$ ways - 1 input degree equal to 2, 1 input degree equal to 1, the rest 0
- m ways - 1 input degree equal to 3, the rest 0

The distribution of incoming edges for groups k_4, k_3, k_2 can be chosen in

- $m(m-1)$ ways - 2 input degrees equal to 1, the rest 0
- m ways - 1 input degree equal to 2, the rest 0

Let $H = (V_H, E_H)$ denote a graph from the class $\mathcal{H}$. Let the indices of the groups be $1 \le k_1 < k_2 < k_3 < k_4 < k_5 < k_6 \le n$, and let one of the scenarios described above be chosen. Then, analogously to the third chapter, we have:

$$\prod_{(u,v)\in E_H} 2\sqrt{uv} \exp\left(O\left(\sum_{v\in V_H} C_H(v)^2/v\right)\right) \sim 2^9 m^9 k_1^{\frac{3}{2}} k_2^{\frac{3}{2}} k_3^{\frac{3}{2}} k_4^{\frac{3}{2}} k_5^{\frac{3}{2}} k_6^{\frac{3}{2}} \tag{B.1}$$

The final step involves summing all probabilities. Here, the main calculations were also performed using a script, which is located in Appendix A. The difference is that there are 10 different scenarios, and we need to sum the results obtained in each of them.

$$\begin{aligned} \mathbb{E}[K_{3,3}] &\sim \sum_{\mathcal{K}_{n,6}^{\text{ord}}} \frac{(m^2-2)^2(m^2-1)^2(5m^4-5m^2+4)}{512m^3(k_1k_2k_3k_4k_5k_6)^{3/2}} \sim \\ &\sim \left(\frac{1}{6!}\right)^{\frac{3}{2}} \sum_{\mathcal{K}_{n,6}} \frac{(m^2-2)^2(m^2-1)^2(5m^4-5m^2+4)}{512m^3(k_1k_2k_3k_4k_5k_6)^{3/2}} = \\ &= \frac{(m^2-2)^2(m^2-1)^2(5m^4-5m^2+4)}{2211840\sqrt{5}m^3} (H_n^{(3/2)})^6 \sim \\ &\sim \frac{(m^2-2)^2(m^2-1)^2(5m^4-5m^2+4)}{2211840\sqrt{5}m^3} \left(\zeta(3/2)\right)^6 \end{aligned} \tag{B.2}$$