Stability number of Markov Random Graphs

Kinga Bagyo
Supervisor: Dr Akshay Gupte

September 2023

Abstract

This paper investigates the stability number of Markov Random Graphs denoted as $G^r_{n,p}$, where the existence of edges is determined by a Markov process governed by a decay parameter $r \in (0, 1]$. The primary objective is to calculate the exact stability number for various Markov Random Graphs and compare it to existing lower bounds. The study employs alternative lower and upper bounding techniques, including Greedy algorithms and spectral lower bounds, and analyzes their computational efficiency and accuracy. The results reveal that the stability number of $G^r_{n,p}$ is of size $\Omega(\frac{n}{\log(n)})$ and $O(n)$, confirming previous findings by Gupte and Zhu. Additionally, the paper extends the research on independence number to the chromatic number of Markov Random Graphs and explores the connection of these graph characteristics to the Lovász ϑ-function, which is to be implemented using semidefinite programming (SDP) during the Year 4 Mathematics Project.

1 Introduction

Consider an undirected graph G, comprising a vertex set V and an edge set E. Any set of vertices that are pairwise non-adjacent is referred to as a stable set (or independent set). For a given graph G, the largest possible number of vertices that can be included in a stable set is termed the stability number (or independence number) of the graph and is denoted by $\alpha(G)$. Similarly, a set of vertices that are all pairwise adjacent form a clique in G, and the maximum cardinality of a clique is the clique number of G, denoted by $\omega(G)$. Computing $\alpha(G)$ (or $\omega(G)$) exactly, commonly known as the Maximum Stable Set Problem (MSSP) (or the Maximum Clique Problem (MCP), respectively) are strongly NP-hard combinatorial problems. Although this report focuses on the MSSP, we must note that these problems are related by the equality

$$\alpha(G) = \omega(\bar{G}).^1 \tag{1}$$

Many research papers were published on approximating these values for various graphs, both theoretically and computationally through optimization algorithms. As explained by Gupte and Zhu [5], Graphs generated using randomization methods are well-suited for exploring a wide range of structural graph properties, but so far most analysis has focused on the classical binomial random graph, often referred to as the Erdős-Rényi-Gilbert random graph. It is denoted by $G_{n,p}$, where n represents the number of vertices

[1] Galli and Letchford [4], p.159.

and each edge has a given fixed probability p of being present in the graph. There is rich history on bounding $\alpha(G_{n,p})$ asymptotically, and research has shown that the largest independent sets in $G_{n,p}$ are typically of size $\Theta(\log(n))$ with high probability (w.h.p.).[2]

Gupte and Zhu [5] arge that "this classical model does not capture any dependency structure between edges that can appear in real-world networks". Thus, a new class of random graphs is defined in their paper called Markov Random Graphs and denoted $G^r_{n,p}$, "whose existence of edges is determined by a Markov process that is also governed by a decay parameter $r \in (0, 1]$." This is explained in detail in section 2.

This project builds upon the results and findings established by Gupte and Zhu [5], who derived both lower and upper boundaries for the independence number of $G^r_{n,p}$. Nevertheless, the question whether their lower bound of size $\Omega(\frac{n}{\log(n)})$ is tight is left open[3], hence the main aim of this study is to calculate the exact stability number for a range of different Markov Random Graphs and compare it to the mentioned lower bound. Additionally, it employs alternative lower and upper bounding techniques from different research papers, utilizing Python for implementation, to enable comparative analysis within this question.

2 Graph generation

Quoting Definition 1 from Gupte and Zhu [5], for any $p, r \in (0, 1]$, a Markov random graph $G^r_{n,p}$ is a random graph on n vertices $v_1, ..., v_n$ wherein the edge probabilities are defined as follows:

1. for $2 \leq i \leq n$, edge (v_1, v_i) exists with probability p,
2. for $2 \leq j < i \leq n$, probability of edge (v_j, v_i) depends on whether edge (v_{j-1}, v_i) is present or not, and is independent of all other edges, and we have the following dependency structure on their conditional probabilities,

$$\mathbb{P}\{\text{edge } (v_{j+1}, v_i) \text{ exists} \mid \text{edge } (v_j, v_i) \text{ does not exist}\} = \mathbb{P}\{\text{edge } (v_j, v_i) \text{ exists}\}$$
$$\mathbb{P}\{\text{edge } (v_{j+1}, v_i) \text{ exists} \mid \text{edge } (v_j, v_i) \text{ exists}\} = r\ \mathbb{P}\{\text{edge } (v_j, v_i) \text{ exists}\}.$$

Adhering to the original paper's terminology, we will keep calling the parameter p as the initial probability, and the parameter r as the decay parameter "since it decreases the edge probability by this factor whenever a previous edge is present". Note that $G^1_{n,p}$ is isomorphic to $G_{n,p}$, hence Markov random graphs are a generalization of the Erdős-Rényi-Gilbert model.

For the purposes of this project, we have generated 500 random graphs in Python[4] with all possible combinations of the following values of n. p and r:

- $n \in [20, 40, 60, 80, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]$,
- $p \in [0.05, 0.25, 0.5, 0.75, 0.95]$ and
- $r \in [0.05, 0.25, 0.5, 0.75, 0.95]$.

Note that these graphs become sparser as r decreases (as well as p).

[2] Gupte and Zhu [5], p.1.
[3] Gupte and Zhu, Theorem 1.1.
[4] See Appendix, Figure 8 for code.

3 Stability number

Computing $\alpha(G)$ can be done by solving the following optimization problem.

$$\begin{aligned} \alpha(G) = &\max \sum_{i=1}^{n} (x_i) \\ &\text{s.t. } x_i + x_j \leq 1 \quad \forall i, j \in E(G) \\ &x \in \{0,1\}^n \end{aligned} \tag{2}$$

First, the stability number of the generated graphs was attempted to be calculated by solving equation 2 using a branch and bound (B&B) algorithm, the mixed integer linear programming solver (`milp`) from the `scipy.optimize` package[5]. As expected, it performed very well for sparse graphs, but e.g. in the case $r = 0.95$, computing the stability number of graphs on more than 80 nodes became very computationally expensive. Thus, the stability number of the remaining graphs with more nodes was calculated with Gurobi Optimization[6], which is the fastest solver in the world currently.[7] Yet, the values of the average $\alpha(G^r_{n,p})$ were still only found up to $n = 100$ for $r = 0.95$ and to $n = 300$ for $r = 0.75$. Unfortunately, trying to find the clique number of the complement using the built-in solver of the `networkx` package also proved to be inefficient.

The results show in Figure 1 confirm our expectations in that the stability number decreases as r increases, and to some extent it also decreases as p grows. Clearly, if r is close to 1, the reduction in probability is not that significant, or if p is large, there is high probability that an edge exists between two vertices, implying that the size of the maximal independent set is small. Yet, the effect of p on the stability number is much less significant than that of r as the average $\alpha(G^r_{n,p})$ appears to depend on p only for small n. As n grows, the average stability number converges to the same value for all p (for any fixed n and r). This implies that p becomes irrelevant as $n \to \infty$. Furthermore, we can observe it in Figure 1 that the smaller r is, the faster the stability number for different values of p converge. Hence, it makes sense to consider the asymptotic behaviour of the stability number in relation to r only[8]

Gupte and Zhu [5] proved that w.h.p. $G^r_{n,p}$ has independent sets of size at least $\frac{1-r}{2+\epsilon}\frac{n}{\log(n)}$ for arbitrary $\epsilon > 0$ and sets of size at most $(e^{-r} + \frac{r}{10})n$. These lower and upper bounds are also illustrated in Figure 1. The actual value of the lower bound compared to the exact independence number is clearly very poor, but its rate of increase is rather close to the tendencies of $\alpha(G^r_{n,p})$.

Although we can only make estimations based on the numerical data obtained for $n \leq 1000$ and it would be unreliable to extrapolate for values larger than that since the independence number may exhibit different tendencies for n beyond 1000, based on Figure 1 the growth of $\alpha(G^r_{n,p})$ appears quasi-linear in n for all r, p. Certainly, because of the decay parameter r, Markov random graphs are sparser than Erdős-Rényi-Gilbert graphs, therefore it makes intuitive sense that if $\alpha(G_{n,p})$ is of size $\Theta(\log(n))$, then $\alpha(G^r_{n,p})$ is larger and grows faster. The stability numbers calculated clearly confirm the results of Gupte and Zhu [5] that $\alpha(G^r_{n,p})$ is of size $\Omega(\frac{n}{\log(n)})$ and $O(n)$.

[5]See Appendix, Figure 10 for code.

[6]See Appendix, Figure 11 for code.

[7]Find all data obtained of stability numbers and lower bounds on GitHub in the Appendix.

[8]Gupte and Zhu also provided asymptotic lower and upper bounds only in terms of n and r.

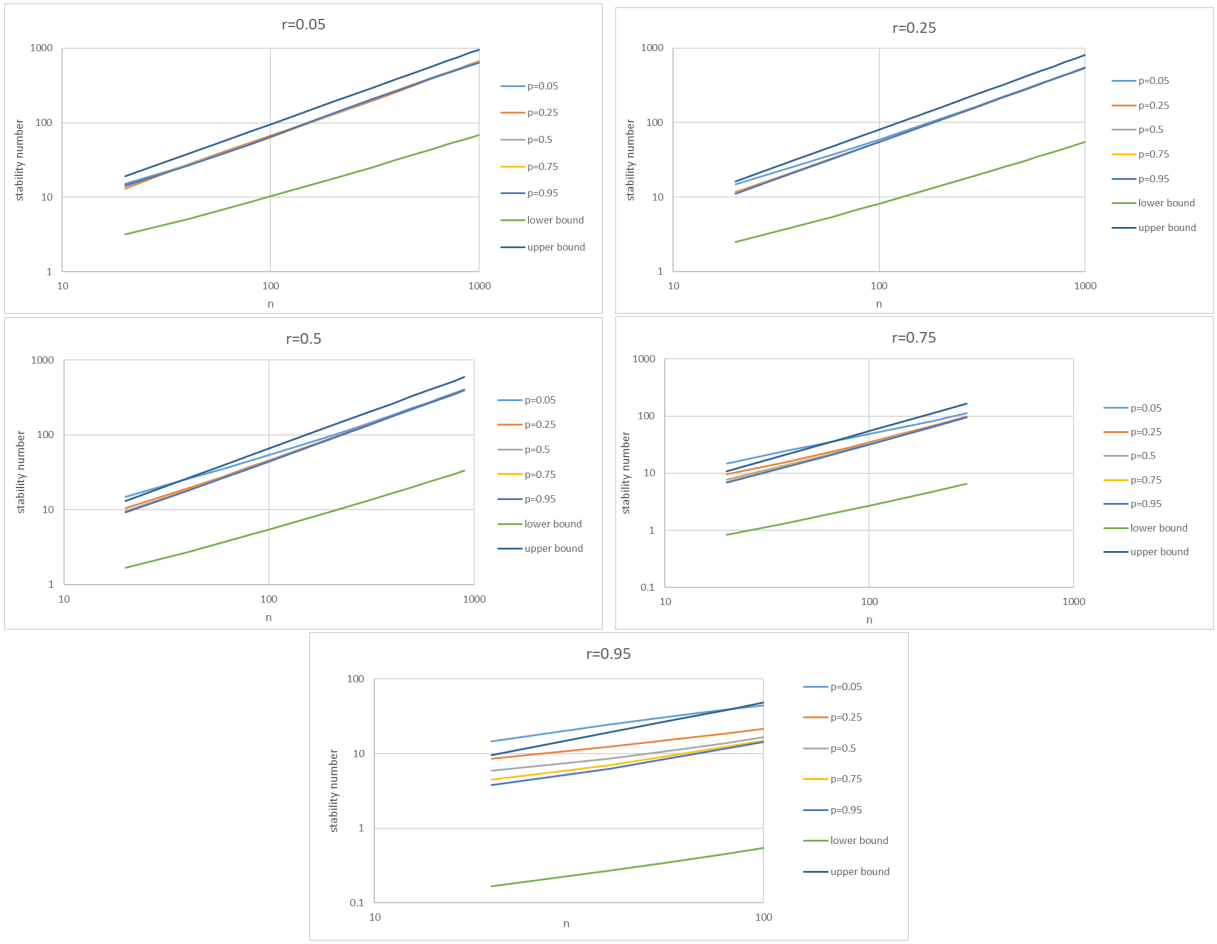
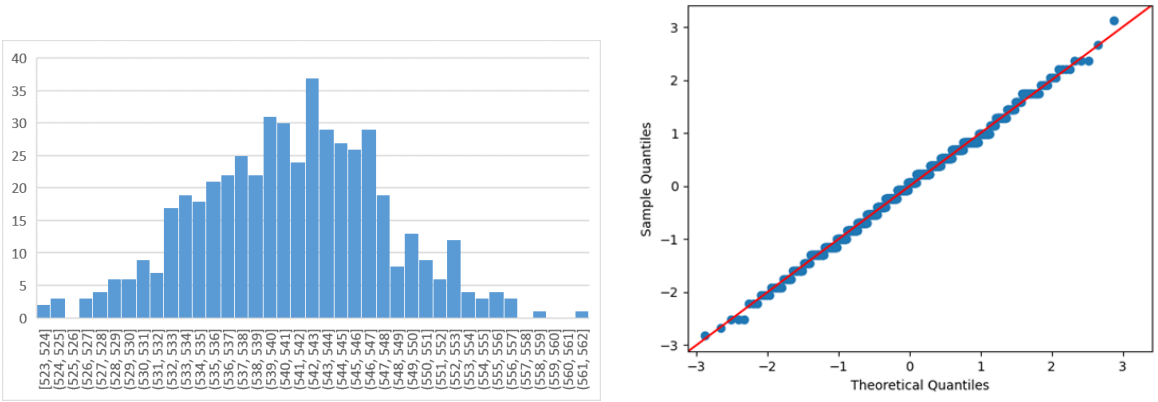


Figure 1: Average stability number of 500 randomly generated Markov random graphs for different values of r and p. Lower and upper bounds included based on the paper by Gupte and Zhu.

Moreover, judging by the tendencies of the lower an upper bounds illustrated, it is likely that a function $f(n) = \Omega(n/\log(n))$ exists such that $f(n) = O(n)$ and $\alpha(G^r_{n,p}) = \Theta(f(n))$ as $\alpha(G^r_{n,p})$ grows somewhat faster than $\frac{n}{\log(n)}$ while the upper bound of size $\Theta(n)$ remains tight as n grows. It is difficult to draw a conclusion due to the quasi-linear nature of the function $\frac{n}{log(n)}$, but it is also possible that for small values of r the stability number is closer to $\Theta(n)$, while for larger values of r, it tends towards $\Theta(\frac{n}{\log(n)})$.

To examine the distribution of the values of $\alpha(G^r_{n,p})$ found, histograms and quantile-quantile plots were produced for several combinations of n, p and r[9]. All of these are very much like the example case shown in Figure 2, the histograms follow the shape of a bell curve and Q-Q plots fall along the 45° straight line implying normal distribution. All of the 500 values generated for all combinations of n, p and r fall between the lower and upper bounds given by Gupte and Zhu.

[9]All data on the distribution of the stability numbers can also be found on GitHub



(a) Histogram (b) Q-Q Plot

Figure 2: Histogram and Quantile-Quantile Plot showing that stability numbers are normally distributed. The case of $r = 0.25$, $p = 0.75$, $n = 1000$ is illustrated, but all other combinations of n, p and r have similar graphs.

4 Approximations

Since the stability number is so computationally expensive to find, we could also try approximating it by lowering tolerance for the optimal solution in the optimization solver, setting a maximum iteration number or setting a time limit for the solver. However, it is also NP-hard to approximate alpha within a factor arbitrarily close to $|V|$. For this reason, we are going to consider other methods to bound α for Markov random graphs from above and below in the hopes of finding some more accurate and computationally efficient bounds.

5 Lower Bounds

5.1 Original Lower Bound

Since this lower bound is independent of p, clearly the performance ratio is best for $p = 0.95$ (since this is the case that yields the smallest average stability number). As discussed before, it is poor as an exact lower bound, but it is a decent lower bound on the rate of $\alpha(G^r_{n,p})$, especially for larger values of r.

5.2 Greedy algorithm

The Greedy Algorithm is implemented the following way to compute the stability number[10].

Let G be a graph, S be an empty set.

1. *Add v_1 to S.*
2. *If v_2 is not connected to v_1, add it to set S.*

[10]See Appendix, Figure 12 for code.

3. *Add all the subsequent vertices to the set S unless they are connected to any of the existing elements in S by an edge.*

However, merely using this approach proved to be completely inefficient since it starts including the vertices with the highest degree in the stable set, so we first sorted the vertices in ascending order based on vertex degrees. This did not require much higher computational cost, but yielded very significant improvements in the lower bound. There could have been a possibility for further improvement by considering the so-called "secondary vertex degrees" of the vertices in the cases where two vertices had identical degree, as proposed by Ballard-Myer [1]. The secondary vertex degree of a vertex v is defined as the sum of the degrees of each vertex adjacent to v and denoted $\deg^2(v)$. If two vertices u and v have the same degree, but $\deg^2(u) \geq \deg^2(v)$, then it is more preferable to choose vertex u to be in the stable set S, since both u and v eliminate the same number of vertices from being added to S, but the subgraph induced from G by eliminating u and all vertices adjacent to it will be less connected, i.e. it has less edges compared to the subgraph induced by eliminating v and all its adjacent vertices.

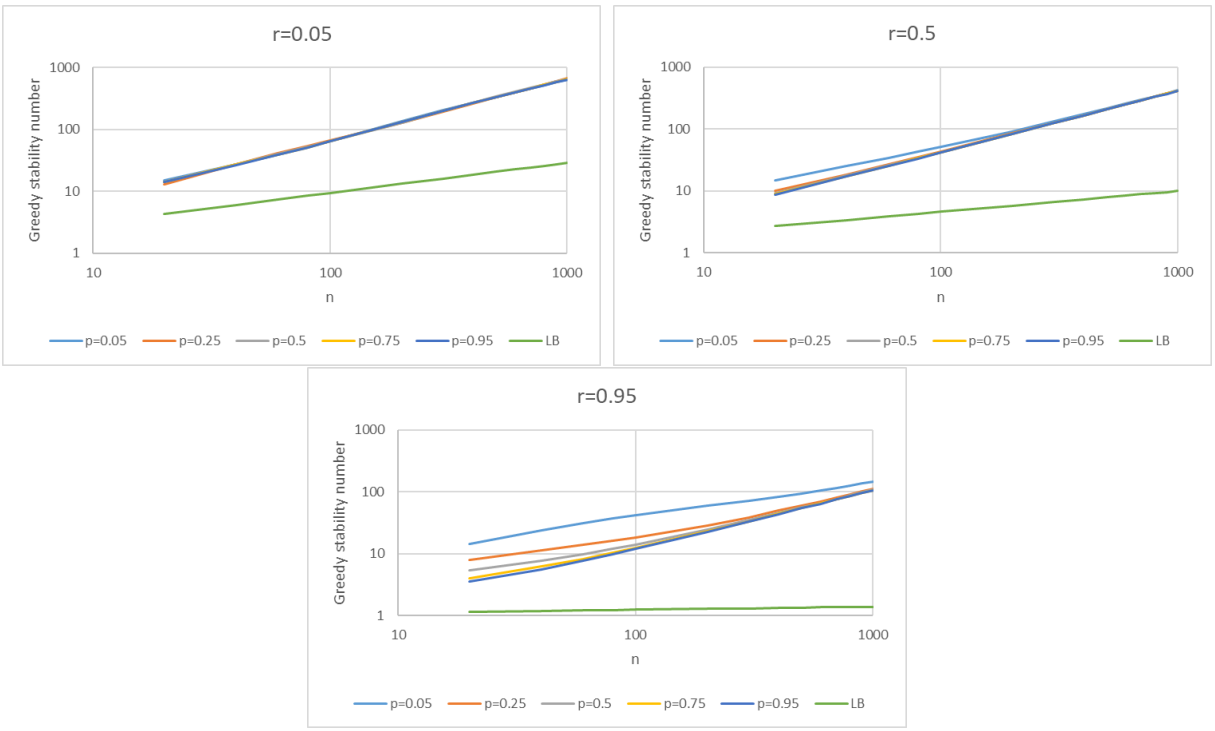


Figure 3: Size of maximal independent set output by the Greedy algorithm with lower bound from Gupte and Zhu.

Gupte and Zhu claim that this version of the Greedy algorithm outputs a set of size $\Omega(n^{\frac{1-r}{2-r}})$. Figure 3 proves that this is most certainly true, but it appears that this lower bound is not very tight, the output grows much faster than $\Theta(n^k)$ for some $0 < k < 0.5$. In fact, the independent set output by the Greedy algorithm also appears to be quasi-linear similar to the growth of the exact value of the MSS, potentially of size $\Theta(n)$.

Gupte and Zhu also deduced that the maximal independent set output by a greedy algorithm has a performance ratio of at most $1 + \frac{\log(n)}{1-r}$ w.h.p. when the lowest degree vertex is picked at each iteration. In this paper, we prefer to keep the performance ratio

between 0 and 1, therefore we graphed the reciprocal of this value as a lower bound in Figure 4. Conducting some superficial analysis using best fit lines, the performance ratio indeed seems to decrease at a rate of $1/\log(n)$. Also, accuracy decreases as r increases (the denser the graphs get), therefore the factor of $(1-r)$ is assumed reasonable. Yet, this estimate seems to be working much better for larger values of r and seems to overestimate the rate of decrease in the performance ratio for smaller values of r.

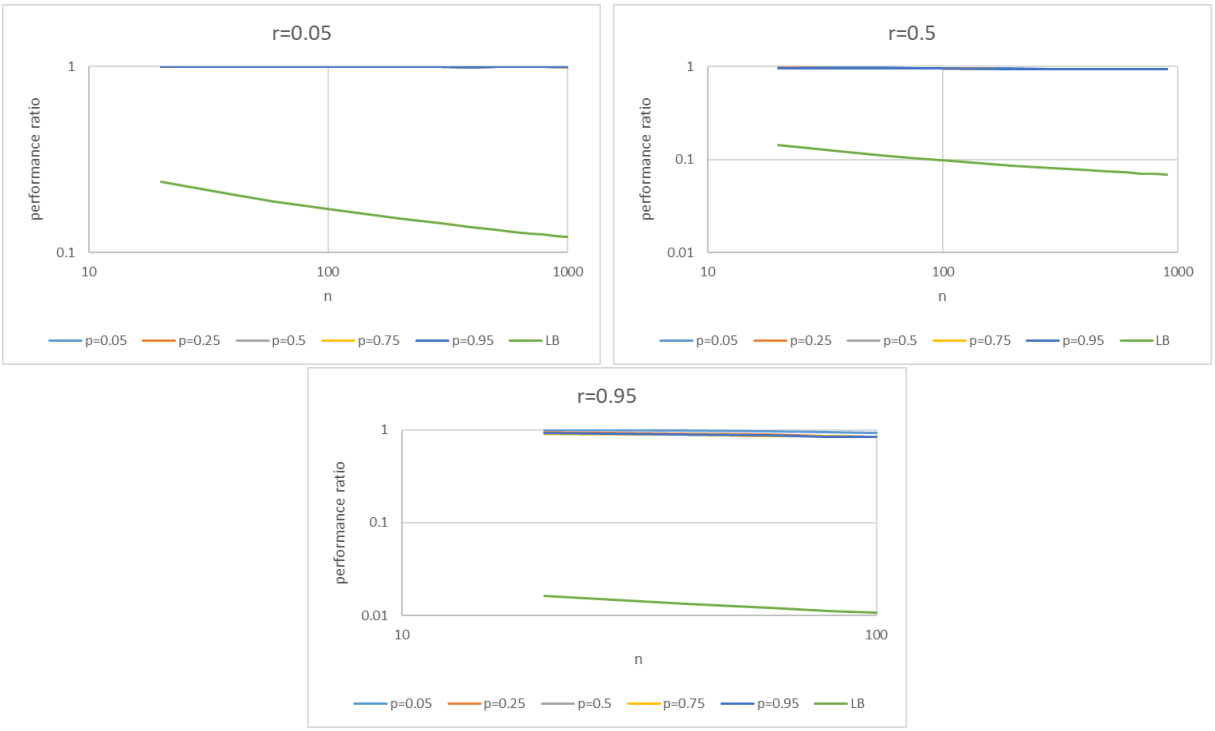


Figure 4: Performance ratio of the Greedy algorithm with lower bound from Gupte and Zhu.

5.3 Spectral Lower Bound

Spectral lower bounds refer to the bounds established for the largest clique number by Yildrim [6] based on the eigenvalues and eigenvectors of the adjacency matrix.

The bound given by Yildrim (26)[11] is defined as follows.

$$\alpha(G) = \omega(\bar{G}) \geq 1 + \frac{2m}{(n - \frac{2m}{n})(\frac{2m}{n} - \lambda_n)}$$

where λ_n is the smallest eigenvalue of $A_{\bar{G}}$, the adjacency matrix of graph $\bar{G}$, n and m are the number of vertices and edges of $\bar{G}$, respectively.

Yildrim (18) relies on the Motzkin-Strauss formulation of the MCP[12]:

$$1 - \frac{1}{\omega(G)} = \max_{x \in \Delta_n} x^T A_G x$$

where Δ_n denotes the $(n-1)$ dimensional unit simplex in $\mathbb{R}^n$, i.e $\Delta_n := \{x \in \mathbb{R}^n : e^T x = 1, x \geq 0\}$. The full formulation of the problem is rather long, but it is available in the original paper.

[11] See Appendix, Figure 13 for code.
[12] See GitHub for code.

5.4 Efficiency of Lower Bounds

5.4.1 Accuracy

As shown in Figure 5, all lower bounds appear to be performing gradually worse as n increases, confirming our expectations, however, the decline in performance ratio is the sharpest in the case of the spectral lower bounds. There is very little difference between the bounds yielded by Yildrim (18) and (26), but (18) is slightly better in all cases (at the cost of much larger computational times). The Greedy algorithm is predominantly and significantly better in terms of performance ratio and its comparatively slow rate of decrease in accuracy. The Greedy lower bound is particularly precise for small values of r when the graph is sparse. Interestingly, however, the original lower bound proposed by Gupte and Zhu expressed in a simple formula that does not require any calculations with matrices appears to perform more consistently compared to the spectral bounds, therefore it is likely that it performs better than Yildrim (18) and (26) for large values of n (especially for larger values of r). Yet, for very small r, the accuracy of the bound by Gupte and Zhu decreases just as fast those by Yildrim.

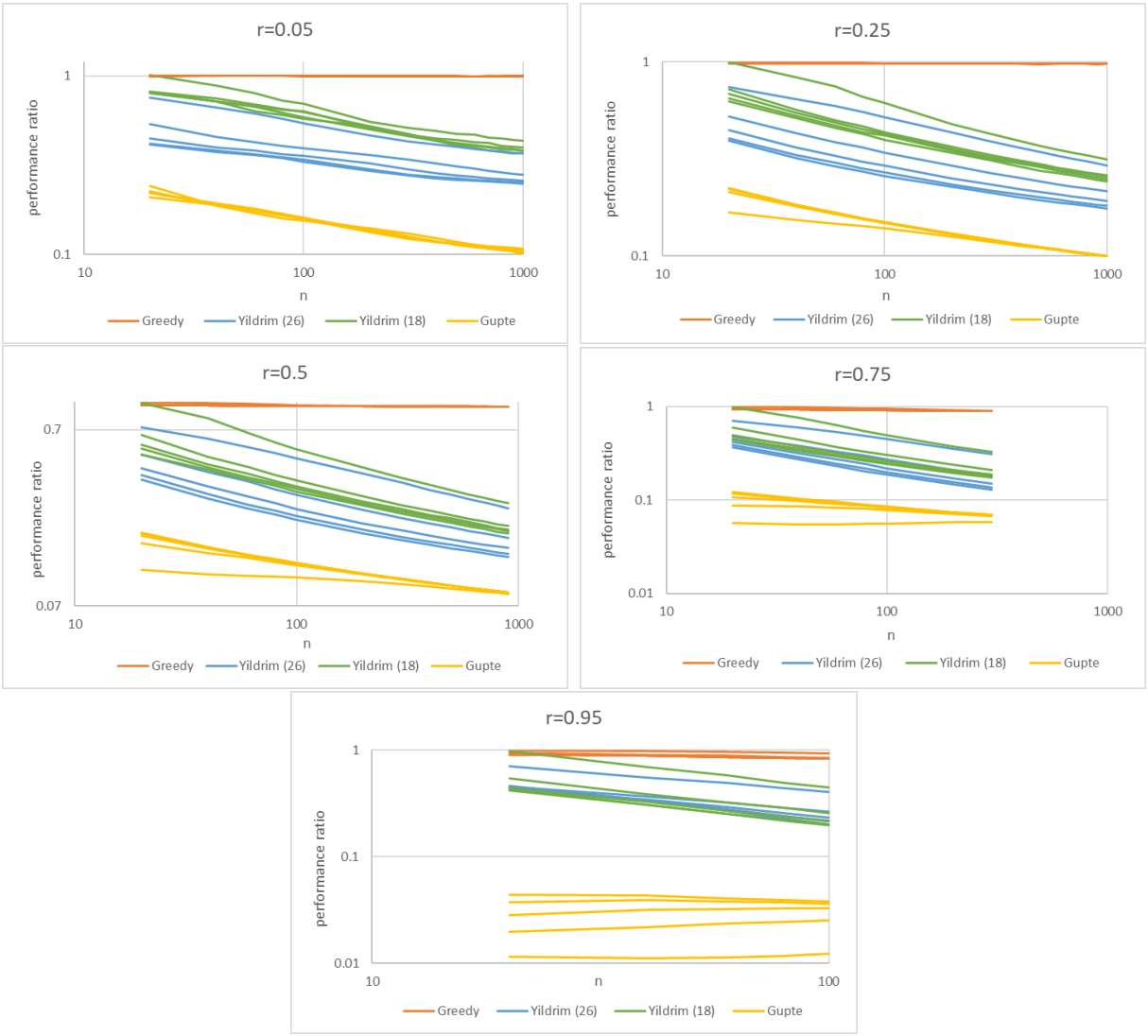
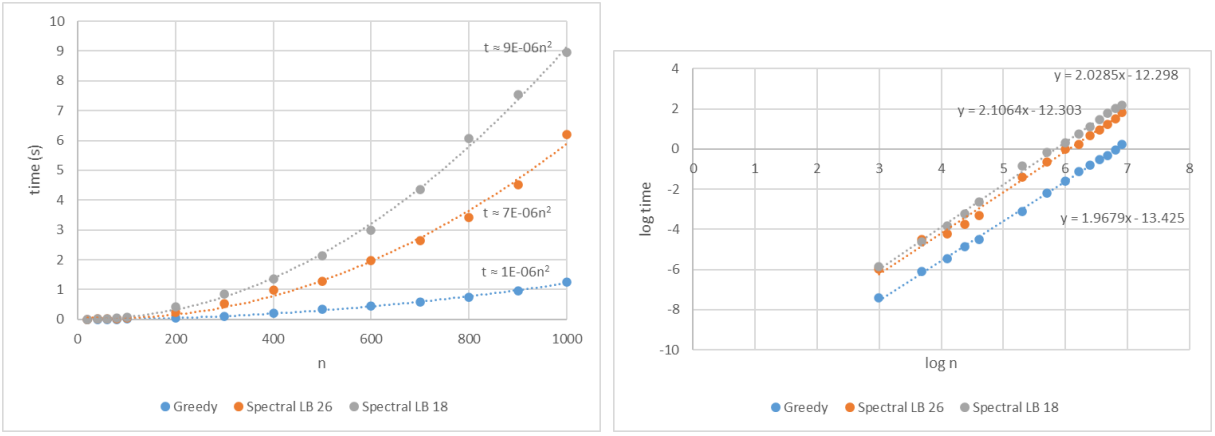


Figure 5: Performance ratio of different lower bounds.

5.4.2 Computational Time

It appears that for a fixed value of n, none of the algorithms require significantly different times to compute a lower bound for the graphs generated depending on the values of p and r. Thus, it appears that computational time is only a function of n. All algorithms were run for a selection of at least 50 different graphs and the average output time was recorded[13]. Figure 6 shows how the computational cost increases with n: all algorithms have cost $O(n^2)$, with the Greedy algorithm being most efficient, requiring only $\frac{1}{9}$th of the time of Yildrim (18) and $\frac{1}{7}$th of the time of Yildrim (26).



(a) (b)

Figure 6: Computational times of different lower bound algorithms.

5.5 Upper Bounds

5.5.1 Original Upper Bound

As mentioned before, the upper bound introduced by Gupte and Zhu is relatively tight, it is clear that $\alpha(G^r_{n,p})$ is of size $O(n)$.

5.5.2 Lovász ϑ-function

The Lovász ϑ-function is a real-valued function associated with a given graph that provides valuable insights into its structural properties. Specifically, the Lovász theta function serves as an upper bound for the stability number, and simultaneously offers a lower bound on the clique cover number, the minimum number of cliques needed to cover all vertices of the graph denoted by $\bar{\chi}(G)$:

$$\alpha(G) \leq \vartheta(G) \leq \bar{\chi}(G). \tag{3}$$

Note that $\bar{\chi}(G) = \chi(\bar{G})$ where $\chi(\bar{G})$ is the chromatic number of the complement of G, i.e. the least number of colours required to colour the vertices of the graph $\bar{G}$ without any two adjacent vertices having the same colour. Hence, we can also write

$$\omega(G) \leq \vartheta(\bar{G}) \leq \chi(G). \tag{4}$$

[13]All data recorded can be found on GitHub in the Appendix

Determining the exact value of the Lovász theta function can be computationally challenging for large graphs, but semidefinite programming (SDP) relaxations and tightenings have been developed to efficiently approximate it. SDP relaxations transform the original integer optimization problem into a continuous one. Specifically, in the context of the Lovász theta function, SDP relaxations reframe the problem by utilizing positive semidefinite matrices to encode vertex relationships within the graph. These relaxations offer a practical approach to compute an approximate $\vartheta(G)$ efficiently.

Moreover, SDP relaxations can be further refined (tightened) to yield more precise approximations of $\vartheta(G)$. This refinement process involves introducing constraints, often referred to as cutting planes[14]. Various techniques and heuristics can enhance the precision of the relaxation, resulting in tighter bounds on graph stability numbers and clique cover numbers. However, it's worth noting that adding cutting planes can significantly increase computational time.

The SDP formulation of the Lovász function involving tightened relaxations from Gaar, Siebenhofer and Wiegele [3] applied to Markov random graphs will be part of the Year 4 Project. Moreover, Dukanovic and Rendl [2] claim that the SDP formulation of the Lovász ϑ-function can be tightened toward either $\chi(G)$ or $\omega(G)$ by adding several types of cutting planes, which might also yield an even tighter upper bound on $\alpha(G)$ than that of Gupte and Zhu, possibly of size $o(n)$, and answers the question posed in their paper.

6 Chromatic number

As mentioned before, the Lovász function connnects the chromatic number and the independence number by providing a simultaneous bound on these values, but they are also related by the following inequalities:

$$n \leq \alpha(G)\chi(G) \tag{5}$$

$$\chi(G) \geq \omega(G) = \alpha(\bar{G}) \tag{6}$$

Furthermore, by searching for$\chi(G)$, we partition all vertices of the graph G into the smallest possible number of disjoint independent sets (i.e. chromatic partitioning). Hence, finding the chromatic number of Markov random graphs is a natural extension of this project on their stability number. This will be carried out as part of the Year 4 Mathematics Project as well as the SDP formulation of the Lovász ϑ-function.

One algorithm proposed by Ballard-Myer [1] based on iteratively finding the stability number is as follows.

Let G be a graph.

1. *Find the maximum stable set (MSS) of G, denote it M_1.*
2. *Find the MSS of $G \setminus M_1$, denote it M_2.*
3. *Continue finding the MSS of $G \setminus (M_1 \cup ... \cup M_s)$ until all vertices belong to some M_i for $i \in [1, 2, ..., k]$.*

[14]One approach to this is by Galli and Letchford [4].

Although it was shown that k, the number of maximum stable sets extracted, does not necessarily equal the exact value of the chromatic number, at least k provides a promising lower bound on $\chi(G)$.

We could also implement the Greedy algorithm to provide another lower bound on the chromatic number. In this case, we the following strategy was used in Python[15]:

Let G be a graph, and $c_1, c_2, ..., c_k, ..., c_n$ be the available colours.

1. *Assign colour c_1 to v_1.*
2. *Assign colour c_2 to v_2 if edge (v_1, v_2) exists, otherwise assign colour c_1 to v_2.*
3. *In case of v_3, exclude the colour assigned to v_j for all $j < 3$ from the list of available colours if edge (v_j, v_3) exists. Assign the smallest indexed available colour to v_3.*
4. *Continue assigning colours to all vertices in a similar fashion to step (3).*

It appears that the chromatic numbers output by this algorithm are of size $\Theta(\log(n))$, as shown below in Figure 14. Yet, it remains to show if sorting the vertices in increasing or decreasing order based on vertex degrees would improve the chromatic number output by the Greedy algorithm. Similar to the analysis given for the stability number in this report, the accuracy and computational efficiency of different lower and upper bounds for the chromatic number will also be incorporated in the Year 4 project.

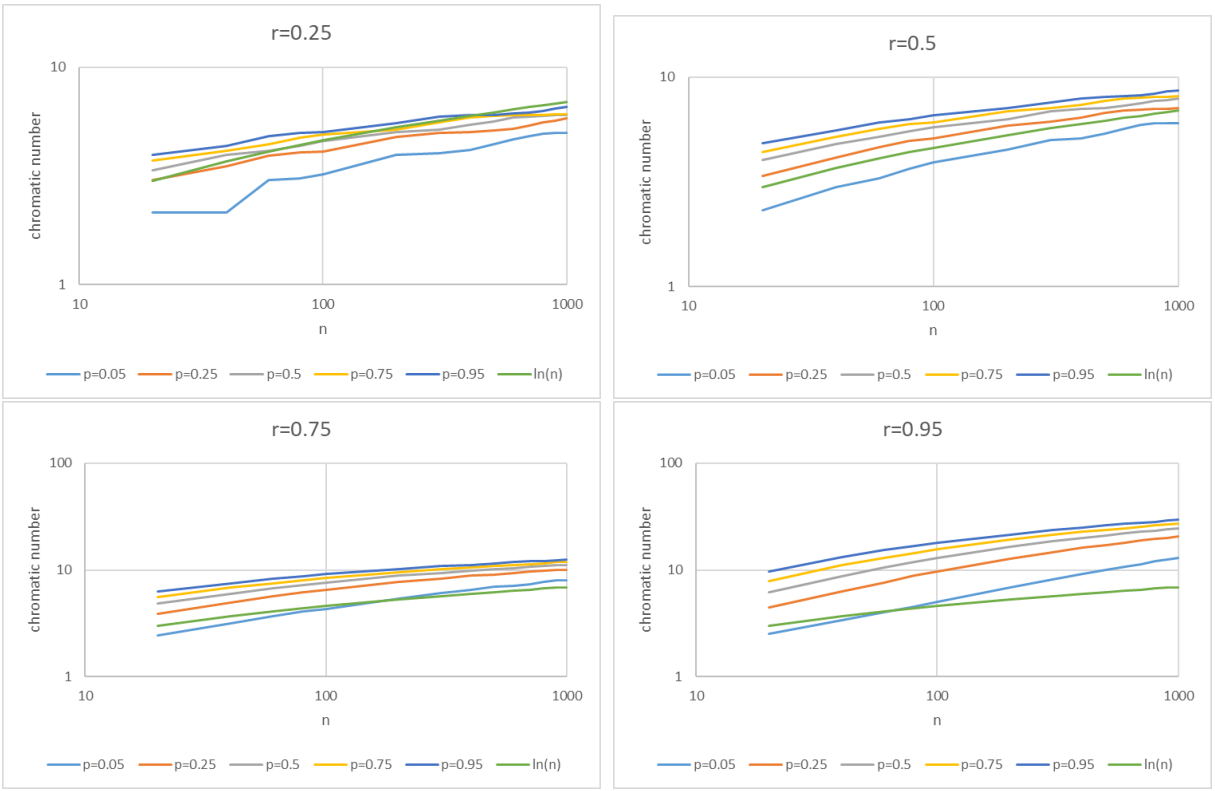


Figure 7: Average chromatic number output by the Greedy Algorithm.

[15]See Appendix, Figure 14 for code

Appendix

GitHub Repository containing all code with comments and results obtained: `https://github.com/bagyokinga/Markov-Random-Graphs`

```
import numpy as np

def generate_random_graphs(n,p,r):
    # Set up edge probability matrix
    P = np.zeros((n,n))

    # Set up adjacency matrix (upper triangular)
    A = np.zeros((n,n))

    # Set probability of edge (1,i) to p for all i > 1
    P[0] = np.ones(n) * p
    P[0,0] = 0

    # Generate Markov random matrix
    for i in range(n):
        for j in range(i, n):
            if np.random.rand() < P[i,j]:
                A[i,j] = 1
                # Apply reduction factor if edge exists
                if i < n-1 and j > i+1: P[i+1, j] = r * P[i,j]
            else:
                # Same probability if edge DNE
                if i < n-1 and j > i+1: P[i+1, j] = P[i,j]

    return A
```

Figure 8: Function to generate Markov Random Graphs.

```
import numpy as np

def edge_matrix(A):
    n = len(A)        # number of vertices

    # Count number of edges
    m = 0
    for i in range(n):
        for j in range(i+1, n):
            if A[i,j] == 1:
                m += 1

    # Create edge matrix from A
    E = np.zeros((m,n))
    count = 0
    for i in range(n):
        for j in range(i+1,n):
            # If there is an entry of 1 in A,
            # add a row to E to represent an edge
            if A[i,j] == 1:
                E[count, i] = 1
                E[count, j] = 1
                count += 1

    return E
```

Figure 9: Function to create an edge matrix whose rows correspond to the edges of the graph.

```
import numpy as np
from scipy.optimize import LinearConstraint
from scipy.optimize import Bounds
from scipy.optimize import milp

def alpha(E):
    n = len(E[0])          # number of vertices in the graph
    m = len(E)             # number of edges in the graph

    # MILP optimization
    c = -np.ones(n)

    # For each edge (i,j), we need x_i + x_j <= 1
    b_u = np.ones(m)
    b_l = np.zeros(m)

    constraints = LinearConstraint(E, b_l, b_u)
    bounds = Bounds(lb=0, ub=1)      # 0 <= x_i <= 1
    integrality = np.ones_like(c)

    # Define milp to be optimized
    res = milp(c=c, constraints=constraints, bounds=bounds,
               integrality=integrality)

    a = int(np.sum(res.x))

    return a
```

Figure 10: Function to find the stability number using `scipy`.

```
import numpy as np
import gurobipy as gp
from gurobipy import GRB

def alpha_gurobi(E):
    n = len(E[0])          # number of vertices in the graph
    m = len(E)             # number of edges in the graph

    # Create model
    model = gp.Model("MarkovGraphs")
    model.Params.LogToConsole = 0

    # Create variables
    x = model.addMVar(shape=n, vtype=GRB.BINARY, name="x")

    # Set objective
    obj = np.ones(n)
    model.setObjective(obj @ x, GRB.MAXIMIZE)

    # Build RHS vector
    rhs = np.ones(m)

    # Add constraints
    model.addConstr(E @ x <= rhs, name="c")

    # Optimize model
    model.optimize()

    a = np.sum(x.X)

    return a
```

Figure 11: Function to find the stability number using `gurobipy`.

```
import numpy as np

def Greedy(A):
    n = len(A[0])           # number of vertices of the graph

    # Make A symmetric
    for i in range(1,n):
        for j in range(i):
            A[i,j] = A[j,i]

    # Add up the rows of A to get the vertex degrees
    vertex_degs = np.zeros(n)
    for i in range(n):
        vertex_degs[i] = np.sum(A[i])

    # Sort vertices by vertex degrees
    vertices = np.array(range(n))
    sorted_vertices = []

    for i in range(n):
        j = np.argmin(vertex_degs)
        sorted_vertices.append(vertices[j])
        vertex_degs[j] = n

    # Add next vertex to set if it is not connected to
    # any of the vertices in set
    indep_set = []
    for i in sorted_vertices:

        # Create a variable that determines if next vertex is
        # independent from the existing elements of the set
        isIndep = True
        for v in indep_set:
            if A[v,i] == 1:
                isIndep = False

        # if vertex is independent, add it to the set
        if isIndep == True:
            indep_set.append(i)

    return len(indep_set)
```

Figure 12: Function to find a lower bound for the stability number using the Greedy Algorithm.

```
import numpy as np
import networkx as nx

def spectral_lb_26(A):
    n = len(A[0])          # number of vertices of the graph

    # Make A symmetric
    for i in range(1,n):
        for j in range(i):
            A[i,j] = A[j,i]

    # Create adjacency matrix of complement of A,
    # and count the number of edges
    A_comp = np.zeros((n,n))
    m = 0
    for i in range(n):
        for j in range(n):
            if i != j:
                if A[i,j] == 1:
                    A_comp[i,j] = 0
                else:
                    A_comp[i,j] = 1
                    m += 1

    m = m/2        # edges were double counted

    # Check connectivity
    G_comp = nx.from_numpy_matrix(A_comp)
    if nx.is_connected(G_comp) == True:

        # Find the smallest eigenvalue
        min_eigval = np.min(np.linalg.eig(A_comp)[0].real)

        # Return the lower bound given in the paper
        return 1+2*m/((n-2*m/n)*(2*m/n-min_eigval))

    # Return nothing if graph is not connected
    else:
        return None
```

Figure 13: Function to find a lower bound for the stability number using the spectral lower bound defined by Yildrim [6] in eq. (26). Note that The alternative spectral lower bound defined by Yildrim, eq. (18) was also implemented and can be found on GitHub.

```
import numpy as np

def chromatic_Greedy(A):
    n = len(A[0])        # number of vertices in the graph

    # Array denoting colours of vertices
    V = np.zeros(n)
    # Assign colour 1 to vertex 1
    V[0] = 1

    # Assign the 'smallest' available colour to other vertices
    for i in range(1,n):
        possible_colors = list(range(1,n+1))
        # Eliminate colours of previous adjacent vertices
        for j in range(i):
            if A[j,i] == 1:
                if V[j] in possible_colors:
                    possible_colors.remove(V[j])
        V[i] = min(possible_colors)

    return int(np.max(V))
```

Figure 14: Function to find a lower bound for the chromatic number of the graph using the Greedy Algorithm.

References

[1] Joshua C. Ballard-Myer. *Deterministic Greedy Algorithm for Maximum Independent Set Problem in Graph Theory*. 2019. URL: https://www.gcsu.edu/sites/files/page-assets/node-808/attachments/ballardmyer.pdf.

[2] Igor Dukanovic and Franz Rendl. "Semidefinite programming relaxations for graph coloring and maximal clique problems". In: *Mathematical Programming* 109.2-3 (2007), pp. 345–365.

[3] Elisabeth Gaar, Melanie Siebenhofer, and Angelika Wiegele. "An SDP-based approach for computing the stability number of a graph". In: *Mathematical Methods of Operations Research* 95.1 (2022), pp. 141–161.

[4] Laura Galli and Adam N Letchford. "On the Lovász theta function and some variants". In: *Discrete Optimization* 25 (2017), pp. 159–174.

[5] Akshay Gupte and Yiran Zhu. *Large independent sets in Markov random graphs*. 2022. arXiv: 2207.04514 [math.CO].

[6] E Alper Yıldırım. "A simpler characterization of a spectral lower bound on the clique number". In: *Mathematical Methods of Operations Research* 71 (2010), pp. 267–281.