

Tarea 2

Integrantes:

- David Alejandro Alquichire Rincón - dalquichire@unal.edu.co
- Kevin Felipe Marroquín Olaya - kfmarroquino@unal.edu.co
- Tomas David Rodríguez Agudelo - trodrigueza@unal.edu.co

Primer Punto

Con base en el siguiente resultado visto en clase:

Theorem 9.1 Fix integers q and $d \geq 2$ such that $q > 2d$, and consider the problem of counting q -colorings for graphs in which each vertex has at most d neighbors. There exists a randomized polynomial time approximation scheme for this problem.

Let us summarize: The algorithm has \tilde{k} factors Y_j to compute. Each one is obtained using no more than $\frac{48d^3k^3}{\varepsilon^2}$ simulations, and, by (76), each simulation requires no more than $k \left(\frac{2 \log(k) + \log(\varepsilon^{-1}) + \log(8)}{\log\left(\frac{q}{2d^2}\right)} + 1 \right)$ steps of the Gibbs sampler.

The total number of steps needed is therefore at most

$$dk \times \frac{48d^3k^3}{\varepsilon^2} \times k \left(\frac{2 \log(k) + \log(\varepsilon^{-1}) + \log(8)}{\log\left(\frac{q}{2d^2}\right)} + 1 \right)$$

which is of the order $Ck^5 \log(k)$ as $k \rightarrow \infty$ for some constant C that does not depend on k . This is less than Ck^6 , so the total number of iterations in the Gibbs sampler grows no faster than polynomially. Since, clearly, the running times of all other parts of the algorithm are asymptotically negligible compared to these Gibbs sampler iterations, Theorem 9.1 is established. \square

Realice experimentos que permitan dar valores aproximados del número de q -coloraciones de un lattice $\mathbf{k} \times \mathbf{k}$.

Considere $3 \leq k \leq 20$ y $2 \leq q \leq 15$.

(a) Reporte:

- Valores ϵ usados.
- Número de simulaciones (muestras) usadas.
- Números de pasos del "Systematic Gibbs Sampler" usados para cada muestra.

(b) Use algún paquete para el conteo exacto y compárelo con algunos de los resultados obtenidos en el item anterior.

Solución:

Antes de comenzar a solucionar los numerales, daremos una breve descripción del algoritmo que implementaremos, el cual se muestra en la demostración del Teorema 9.1 (Libro *Finite Markov chains and algorithm applications*):

Sea $G = (V, E)$ el grafo reticular $\mathbf{k} \times \mathbf{k}$ con $V = \{v_1, v_2, \dots, v_{n=k^2}\}$ y $E = \{e_1, e_2, \dots, e_{m=2k(k-1)}\}$. Definimos:

$$G_0 = (V, \emptyset)$$

$$G_i = (V, \{e_1, \dots, e_i\})$$

para $1 \leq i \leq m$. Z_i será el número de q -coloraciones del grafo G_i .

Queremos aproximar Z_m , el cual puede ser re escrito como:

$$Z_m = \frac{Z_m}{Z_{m-1}} \times \frac{Z_{m-1}}{Z_{m-2}} \times \dots \times \frac{Z_2}{Z_1} \times \frac{Z_1}{Z_0} \times Z_0.$$

1. Comenzaremos con G_0 , el grafo que tiene todos los vértices (en este caso $\mathbf{k} \times \mathbf{k}$) pero ninguna arista. Esto significa que cualquier asignación de colores es válida, es decir $Z_0 = q^{k^2}$.

Para cada arista que se añade al grafo, estimaremos la proporción $\frac{Z_i}{Z_{i-1}}$. Para esto, podemos notar que las q -coloraciones de G_i son aquellas q -coloraciones de G_{i-1} en las que los vértices de la arista añadida en G_i tienen un color distinto. Por lo tanto, la proporción de coloraciones de G_{i-1} que también son válidas en G_i se convierte en $\frac{Z_i}{Z_{i-1}}$. Es decir, $\frac{Z_i}{Z_{i-1}}$ es la probabilidad de que dos vértices

conectados por la nueva arista tengan diferentes colores. En este sentido:

2. Para cada arista $e_i = \{x_i, y_i\}$, estimamos $\frac{Z_i}{Z_{i-1}}$ realizando múltiples simulaciones en las que, utilizando Gibbs sampler, obtenemos una q -coloración del grafo G_{i-1} , así $\frac{Z_i}{Z_{i-1}}$ será la proporción de simulaciones que resulten en una coloración en la que x_i, y_i tienen colores distintos.
3. Construimos la estimación final multiplicando sucesivamente estas razones, partiendo desde Z_0 (que como vimos es trivial de calcular).

Realizamos múltiples simulaciones para cada arista añadida, permitiendo que la distribución de q -coloraciones converja adecuadamente realizando suficientes pasos para el Gibbs sampler.

Implementación:

Para representar el retículo, el vértice superior izquierdo será etiquetado como $(1, 1)$ y el inferior derecho como (k, k) , luego una arista entre los vértices (x_1, y_1) y (x_2, y_2) será representada como la pareja $((x_1, y_1), (x_2, y_2))$.

(1,1)	(1,2)	...	(1, k)
(2,1)	(2,2)	...	(2, k)
.	.	.	.
.	.	.	.
.	.	.	.
(k,1)	(k,2)	...	(k, k)

Bibliotecas utilizadas:

```
1 using Random,PlutoUI,Printf, Statistics
```

En primer lugar, implementamos las funciones `create_lattice()`, que inicializa un retículo de $k \times k$ con una coloración aleatoria; y `gen_edges()` que retorna una lista con las aristas del grafo reticular.

```
create_lattice (generic function with 1 method)
```

```
1 function create_lattice(k::Int, q::Int)
2     lattice = zeros(Int, k, k)
3     for i in 1:k
4         for j in 1:k
5             lattice[i, j] = rand(0:q-1)
6         end
7     end
8     return lattice
9 end
```

```
gen_edges (generic function with 1 method)
```

```
1 function gen_edges(k::Int)
2     edges = []
3     for i in 1:k
4         for j in 1:k
5             if i + 1 <= k
6                 push!(edges, ((i, j), (i + 1, j)))
7             end
8             if j + 1 <= k
9                 push!(edges, ((i, j), (i, j + 1)))
10            end
11        end
12    end
13    return edges
14 end
```

A continuación, implementamos el systematic Gibbs sampler mediante la función `gibbs_step()`.

Systematic Gibbs sampler es una variante de Gibbs sampler en la que el vértice v_i se actualiza sistemáticamente en los tiempos $i, n+i, 2n+i, \dots$, donde $n = |V|$. En los tiempos específicos de cada vértice se actualiza el color de dicho vértice de acuerdo a la distribución uniforme sobre el conjunto de colores disponibles (según los vecinos del vértice), a los demás vértices no se les hace cambio alguno. Nosotros iteraremos en orden por cada fila del retículo, luego

$v_1 = (1, 1), v_{k+1} = (2, 1), \dots, v_n = (k, k)$.

v_{-1}	v_{-2}	\dots	v_{-k}
$v_{\{-k+1\}}$	$v_{\{-k+2\}}$	\dots	$v_{\{-2k\}}$
•	•	•	•
•	•	•	•
•	•	•	•
$v_{\{-(k-1)k+1\}}$	$v_{\{-(k-1)k+2\}}$	\dots	$v_{\{-k^2\}}$

```

gibbs_step (generic function with 1 method)
1 function gibbs_step(k::Int, q::Int, lattice::Array{Int,2}, neighbours)
2     used = zeros(Int, q)
3     for i in 1:k, j in 1:k
4         used = zeros(Int, q)
5         unused = []
6         if !isempty(neighbours[i,j])
7             for ne in neighbours[i,j]
8                 used[lattice[ne[1], ne[2]] + 1] = 1
9             end
10        end
11        for color in 0:q-1
12            if used[color + 1] != 1
13                push!(unused, color)
14            end
15        end
16        if !isempty(unused)
17            lattice[i, j] = rand(unused)
18        end
19    end
20 end

```

Note que `gibbs_step()` realiza k^2 pasos del systematic Gibbs sampler.

Ahora, implementamos la función `estimate_ratio()`, en la que realizamos `num_simulations` número de simulaciones y -mínimo- `num_gibbs_steps` número de pasos del systematic Gibbs sampler para estimar $\frac{Z_i}{Z_{i-1}}$. En cada simulación, aumentamos la cuenta de `count` si en la coloración obtenida los vértices de edge ($e_i = (u, v)$), tienen un color distinto.

`estimate_ratio` (generic function with 1 method)

```

1 function estimate_ratio(k::Int, q::Int, num_simulations::Int, num_gibbs_steps::Int,
2     edge, lattice, neighbours)
3     u_y, u_x = edge[1]
4     v_y, v_x = edge[2]
5     count = 0
6     for _ in 1:num_simulations
7         for _ in 1:ceil(num_gibbs_steps/(k*k))
8             gibbs_step(k, q, lattice, neighbours)
9         end
10        if (lattice[u_y, u_x] != lattice[v_y, v_x])
11            count += 1
12        end
13    end
14    ratio = BigFloat(count / num_simulations)
15    return ratio
16 end

```

Finalmente, establecemos los parámetros:

```
k = 5
```



n = 25

m = 40

```
q = 10
```



```
epsilon = 1.0
```



Inicializamos el retículo:

```
lattice = 5×5 Matrix{Int64}:
 0  9  9  7  3
 4  1  6  5  4
 5  4  0  4  4
 4  2  1  9  6
 5  5  7  5  4
1 lattice = create_lattice(k, q)
```

Generamos las aristas del retículo:

```
edges =
▶ [(1, 1), (2, 1)), ((1, 1), (1, 2)), ((1, 2), (2, 2)), ((1, 2), (1, 3)), ((1, 3), (2, 3)),
1 edges = gen_edges(k)
```

Declaramos el valor inicial de z como $Z_0 = q^n$, para esto utilizamos la función BigFloat() de Julia que brinda una precisión arbitraria sobre la variable z (dependiendo de **k** y de **q** puede tomar valores muy grandes):

```
z = 1.0e+25
1 z = BigFloat(q)^n
```

Finalmente, establecemos el número de simulaciones y el número de pasos para el systematic Gibbs sampler, para esto utilizamos los resultados del teorema mencionado anteriormente. Como observación, según dichos resultados se deberían hacer $\frac{48d^3n^3}{\epsilon^2}$. Sin embargo realizaremos $\frac{n^3}{\epsilon^2}$ (manteniendo el orden de n^3 sobre el número de simulaciones) pues de la otra manera para el ejemplo actual de $k = 5$ y $q = 10$ habían pasado más de diez mil segundos (2.7 horas) y aún no se obtenía respuesta.

```

num_simulations = 15625
1 num_simulations = Int(ceil(n^3/epsilon^2))

num_gibbs_steps = 159
1 num_gibbs_steps = Int(ceil(abs(n * ((2 * log(n) + log(1 / epsilon) + log(8)) /
log(q) / 32) + 1)))

1 begin
2 Random.seed!(1234)
3 neighbours = [Vector{Tuple{Int,Int}}]()
for _ in 1:k, _ in 1:k]
4 ratios = []
5 for edge in edges
6   u_y, u_x = edge[1]
7   v_y, v_x = edge[2]
8   ratio = estimate_ratio(k, q, num_simulations, num_gibbs_steps, edge, lattice,
neighbours)
9   Z *= BigFloat(ratio)
10  push!(ratios, BigFloat(ratio))
11  push!(neighbours[u_y, u_x], (v_y, v_x))
12  push!(neighbours[v_y, v_x], (u_y, u_x))
13 end
14 val_est = round(Z)
15 @printf("Número estimado de %d-coloraciones para el retículo %dx%d:\n", q, k, k)
16 @printf("%d\n", val_est)
17 end
    
```

Número estimado de 10-coloraciones para el retículo 5x5:
151236661989137789064263

Para estimar el error del resultado utilizaremos *Mathematica*, que nativamente implementa una función que permite calcular el polinomio cromático P_G de un grafo arbitrario G . Un resultado de teoría de grafos dice que $P_G(q)$ dice el número exacto de q -coloraciones del grafo G .

Nota: En el repositorio de [GitHub](#) en la carpeta */Mathematica*, se encuentra un notebook en el que se realizan los cálculos que se mostrarán en el presente informe, en caso de que un resultado sea obtenido mediante *Mathematica* escribiremos ★.

Sea L_k el grafo reticular $k \times k$, entonces

$$P_{L_5}(10) = 151086899096935604867610 *$$

```
val_real = 151086899096935604867610
```

```
>- val_est = 151236661989137789064263
```

Es decir que la estimación obtenida tiene un error relativo de:

```
1 @printf("%0.2f%%", (abs(val_real-val_est)/val_real)*100)
```

```
>- 0.10%
```

Confirmando que es una muy buena estimación del número de 10-coloraciones del grafo reticular 5×5 .

Algo interesante que podemos notar realizando estos experimentos es que, al menos para $k = 5$ y $q = 10$, las razones $\frac{Z_i}{Z_{i-1}}$ en general son muy cercanas independientemente de la arista e_i :

```
> [0.898816, 0.900736, 0.900416, 0.90016, 0.898432, 0.899008, 0.897792, 0.901824, 0.900672]
```

```
1 ratios
```

```
1 @printf("Desviación estándar de `ratios`: %0.3f", std(ratios))
```

```
>- Desviación estándar de `ratios`: 0.003
```

Es decir que, al menos para este caso, podríamos obtener una estimación del resultado simplemente multiplicando Z_0 por $[\frac{Z_1}{Z_0}]^{|E|}$:

```
approx_z1 = 140227047851122301343831
```

```
1 approx_z1 = BigInt(round(BigFloat(q)^n * ratios[1]^m))
```

```
>- Error relativo: 7.188%
```

Utilizando $\frac{Z_2}{Z_1}$:

```
approx_z2 = 152721727186583571977709
```

```
1 approx_z2 = BigInt(round(BigFloat(q)^n * ratios[2]^m))
```

```
1 @printf("Error relativo: %0.3f%%", (abs(approx_z2 - val_real)/val_real)*100)
```

Error relativo: 1.082%

Utilizando el promedio de ratios:

```
approx_zmean = 151263626104714948938586
```

```
1 approx_zmean = BigInt(round(BigFloat(q)^n * mean(ratios)^m))
```

Error relativo: 0.117%

¿Existe algún resultado sobre esto? Por ejemplo para $k = 9$ y $q = 3$ las aristas horizontales a partir de la segunda fila tienen una razón en general cercana a **0.75** mientras que para el resto de las aristas a **0.666**.

Implementación (C++):

Como mencionamos anteriormente, si hubiésemos realizado $\frac{48d^3n^3}{\epsilon^2}$ simulaciones, probablemente se habría obtenido la respuesta después de muchas horas, tal vez días. Es por esto que decidimos realizar una implementación optimizada del código anterior en el lenguaje C++. Por ejemplo, dejando los mismos parámetros anteriores ($k = 5, q = 10, \epsilon = 1$) y realizando $\frac{48d^3n^3}{\epsilon^2}$ simulaciones el programa en C++ tarda 4463 segundos, lo cual es poco en comparación con las posibles múltiples horas que habría tardado en Julia. Realizando $\frac{n^3}{\epsilon^2}$ a penas se tarda 2.86 segundos versus 19.4 segundos que tardó en Julia.

```
cpp - zsh  cpp - zsh
(4,1) <-> (4,2)
0.901317
186056840750977456777831
(4,2) <-> (4,3)
0.901241
167682060967620038747970
(4,3) <-> (4,4)
0.901244
151122468821583104995129
Estimated number of 10-colorings for 5x5 lattice: 151122468821583104995130

finished in 4463.69 sec
```

Error relativo: 0.024%

```

cpp - zsh
(4,1) <-> (4,2)
0.89792
184947775026897584419578
(4,2) <-> (4,3)
0.906624
167678091585985995576816
(4,3) <-> (4,4)
0.905536
151838548342407414490647
Estimated number of 10-colorings for 5x5 lattice: 151838548342407414490648

finished in 2.85961 sec

```

Error relativo: 0.497%

A continuación mostramos la implementación realizada.

Nota: En la carpeta `cpp` del [GitHub](#) también se encuentra la implementación y detalles sobre la misma.

```

#include <boost/multiprecision/cpp_dec_float.hpp>
#include <boost/multiprecision/cpp_int.hpp>
#include <random>

using namespace std;
using namespace boost::multiprecision;

typedef number<cpp_dec_float<40>> big_float; // floats with 40 digits of precision
typedef cpp_int big_int; // arbitrarily large integers
typedef pair<int, int> pii; // structure to store graph coordinates

// Random number generator
unsigned seed = 1234;
mt19937 rng(seed);

// Initiate lattice with random colors
vector<vector<int>> create_lattice(int k, int q) {
    vector<vector<int>> lattice(k, vector<int>(k));
    for (auto &row : lattice)
        for (auto &cell : row)
            cell = uniform_int_distribution<int>(0, q - 1)(rng);

    return lattice;
}

// Generate lattice edges
vector<pair<pii, pii>> gen_edges(int k) {
    vector<pair<pii, pii>> edges;
    edges.reserve(2 * k * (k - 1));
    for (int i = 0; i < k; i++) {
        for (int j = 0; j < k; j++) {
            if (i + 1 < k) edges.emplace_back(pii{i, j}, pii{i + 1, j});
            if (j + 1 < k) edges.emplace_back(pii{i, j}, pii{i, j + 1});
        }
    }
}
```

```

    }

    return edges;
}

// Systematic Gibbs sampler (k^2 steps)
void gibbs_sampler_step(int k, int q, vector<vector<int>> &lattice, const vector<vector<vector<pii>>> &neighbours) {
    int used[q], unused[q], version = 0, count = 0;
    for (int i = 0; i < k; i++) {
        for (int j = 0; j < k; j++) {
            version++;
            count = 0;
            for (const auto &ne : neighbours[i][j])
                used[lattice[ne.first][ne.second]] = version;

            for (int color = 0; color < q; color++)
                if (used[color] != version)
                    unused[count++] = color;

            lattice[i][j] = (count == 0 ? lattice[i][j] : unused[uniform_int_distribution<int>(0, count-1)(rng)]);
        }
    }
}

// Estimate values of the telescopic product
big_float estimate_ratio(int k, int q, int num_simulations, int num_gibbs_steps, const pair<pii, pii> &edge, vector<vector<int>> &lattice, const vector<vector<vector<pii>>> &neighbours) {
    int u_y = edge.first.first, u_x = edge.first.second;
    int v_y = edge.second.first, v_x = edge.second.second;
    int count = 0;

    for (int i = 0; i < num_simulations; i++) {
        for (int j = 0; j < ceil(num_gibbs_steps/(k*k)); j++)
            gibbs_sampler_step(k, q, lattice, neighbours);
        count += lattice[u_y][u_x] != lattice[v_y][v_x];
    }
}

return big_float(count) / num_simulations;
}

int main() {
    int k, q;
    big_float epsilon;
    cin >> k >> q >> epsilon; // read k, q and eps from input
    int n = k * k; // |V|
    auto lattice = create_lattice(k, q);
    auto edges = gen_edges(k);
    vector<vector<vector<pii>>> neighbours(k, vector<vector<pii>>(k));

    int num_simulations = static_cast<int>(pow(big_float(n), 3) / (epsilon * epsilon));
    int num_gibbs_steps = abs(static_cast<int>(n * ((2 * log(n) + log(1 / epsilon) + log(8)) / log(big_float(q) / 32) + 1)));
    cout << "Sims: " << num_simulations << ", steps: " << num_gibbs_steps << "\n";

    big_float Z = pow(big_float(q), n); // Z = Z_0

    for (const auto &edge : edges) { // edge = {x, y}
        const auto &x = edge.first;

```

```

        const auto &y = edge.second;
        big_float ratio = estimate_ratio(k, q, num_simulations, num_gibbs_steps, edge,
e, lattice, neighbours);
        Z *= ratio;
        // add edge
        neighbours[x.first][x.second].emplace_back(y);
        neighbours[y.first][y.second].emplace_back(x);
    }

    big_int rounded_Z = (Z + 0.5).convert_to<big_int>();
    cout << "Estimated number of " << q << "-colorings for " << k << "x" << k << " lattice: " << rounded_Z << "\n";
} cerr << "\nfinished in " << clock() * 1.0 / CLOCKS_PER_SEC << " sec\n";
}

```

Reporte de resultados:

1 using CSV, DataFrames, Plots, StatsPlots

Para esta sección utilizamos la implementación realizada en C++.

Convención:

- $|k|$ Dimensión retículo.
- $|q|$ Número de colores.
- $|\text{Gibbs}|$ Número de pasos del systematic Gibbs sampler.
- $|\text{Est}|$ Estimación obtenida.
- $|\text{Res}\star|$ Valor real.
- $|\text{R_err}|$ Error relativo (%).
- $|\text{Mean_r}|$ Promedio de las razones obtenidas.
- $|\text{Time}|$ Tiempo en segundos que tardó el programa.

En primer lugar, consideraremos resultados obtenidos con los siguientes parámetros:

$$\text{num_simulations} = \frac{48d^3n^3}{\epsilon^2}$$

$$\text{num_gibbs_steps} = n \left(\left| \frac{2 \log n + \log \epsilon^{-1} + \log 8}{\log \frac{q}{2d^2}} \right| + 1 \right)$$

Nota: Para no extender mucho el documento, solo mostraremos las tablas para el ϵ más pequeño, el resto de datos se podrá visualizar en las gráficas. Todos los resultados obtenidos se encuentran en la carpeta *results* del [GitHub](#).

k=2

$\epsilon = 0.1$

`num_simulations = 19660800`

`file_path_a = "../results/q-colorings/k=2/eps=0.1.csv"`

<code>dfa</code> =	k	q	Gibbs	Est	Res★	R_err	Mean_r	Time
1	2	2	6	2	2	0.0	0.625	1.76
2	2	3	8	18	18	0.0	0.68751	7.76
3	2	4	9	84	84	0.0	0.75693	9.28
4	2	5	11	260	260	0.0	0.80309	12.0
5	2	6	13	630	630	0.0	0.83498	21.3
6	2	7	14	1302	1302	0.0	0.85809	19.7
7	2	8	16	2408	2408	0.0	0.87567	22.4
8	2	9	18	4105	4104	0.02437	0.88936	29.1
9	2	10	20	6569	6570	0.01522	0.90026	42.8
10	2	11	22	10008	10010	0.01998	0.90928	42.6
11	2	12	25	14655	14652	0.02048	0.91688	47.3
12	2	13	27	20752	20748	0.01928	0.92325	44.1
13	2	14	30	28580	28574	0.021	0.92873	48.3
14	2	15	33	38426	38430	0.01041	0.93339	51.7

$\epsilon = 1$

`num_simulations = 196608`

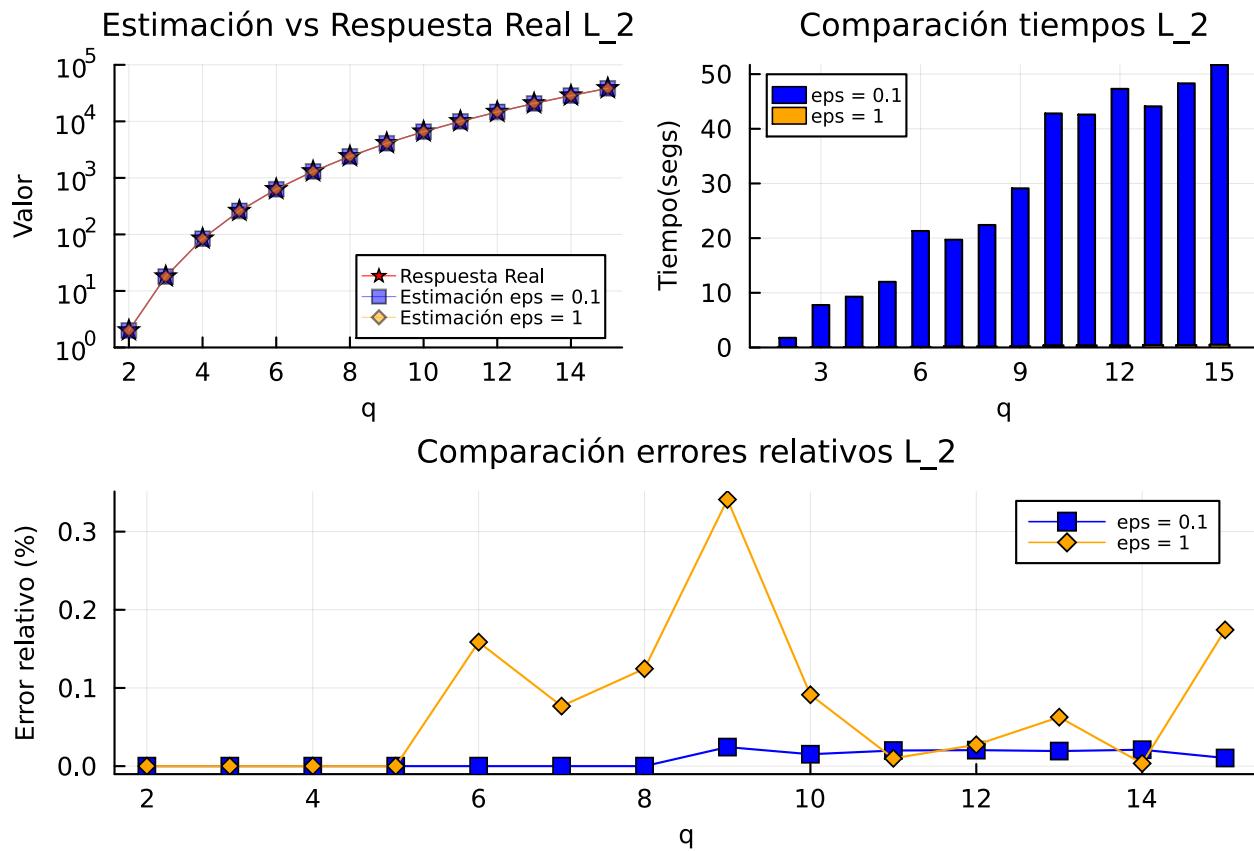
`file_path_b = "../results/q-colorings/k=2/eps=1.csv"`

```
1 dfb = CSV.File(file_path_b, delim=' ', header=true) |> DataFrame;
```

Pasos del Gibbs para $2 \leq q \leq 15$ ($\epsilon = 1$), respectivamente:

► [4, 4, 5, 6, 7, 8, 9, 11, 12, 14, 15, 17, 19, 21]

Visualización de los datos:



Podemos observar como las estimaciones con $\epsilon = 1$ se acercan de muy buena manera a la respuesta teniendo en cuenta que en general las estimaciones con $\epsilon = 0.1$ tardan en promedio 175 veces más.

Nota: La gráfica "Estimación vs Respuesta" está en escala logarítmica (\log_{10}).

$$\mathbf{k=3}$$

$$\epsilon = 0.31$$

$$\text{num_simulations} = 22394879$$

```
file_path_c = "../results/q-colorings/k=3/eps=0.31.csv"
```

dfc	k	q	Gibbs	Est	Res★	R_err	Time
1	3	2	15	2	2	0.0	473.724
2	3	3	19	246	246	0.0	1065.65
3	3	4	24	9618	9612	0.06242	1692.58
4	3	5	27	142803	142820	0.0119	2432.79
5	3	6	31	1166838	1166910	0.00617	3052.65
6	3	7	36	6461619	6464682	0.04738	3529.93
7	3	8	40	27336133	27350456	0.05237	3804.24
8	3	9	45	94983776	95004072	0.02136	5032.08
9	3	10	50	283941823	283982490	0.01432	6117.6
10	3	11	55	754227186	754324670	0.01292	6806.97
11	3	12	60	1822620799	1821684612	0.05139	8302.3
12	3	13	67	4067781228	4067709516	0.001763	8482.63
13	3	14	74	8502754526	8506024982	0.03845	8902.75
14	3	15	74	16824103514	16822697010	0.008361	9452.18

Nota: Estos cálculos se hicieron con una versión no optimizada del código, de ahí los elevados tiempos de ejecución.

$$\epsilon = 1$$

num_simulations = 2239488

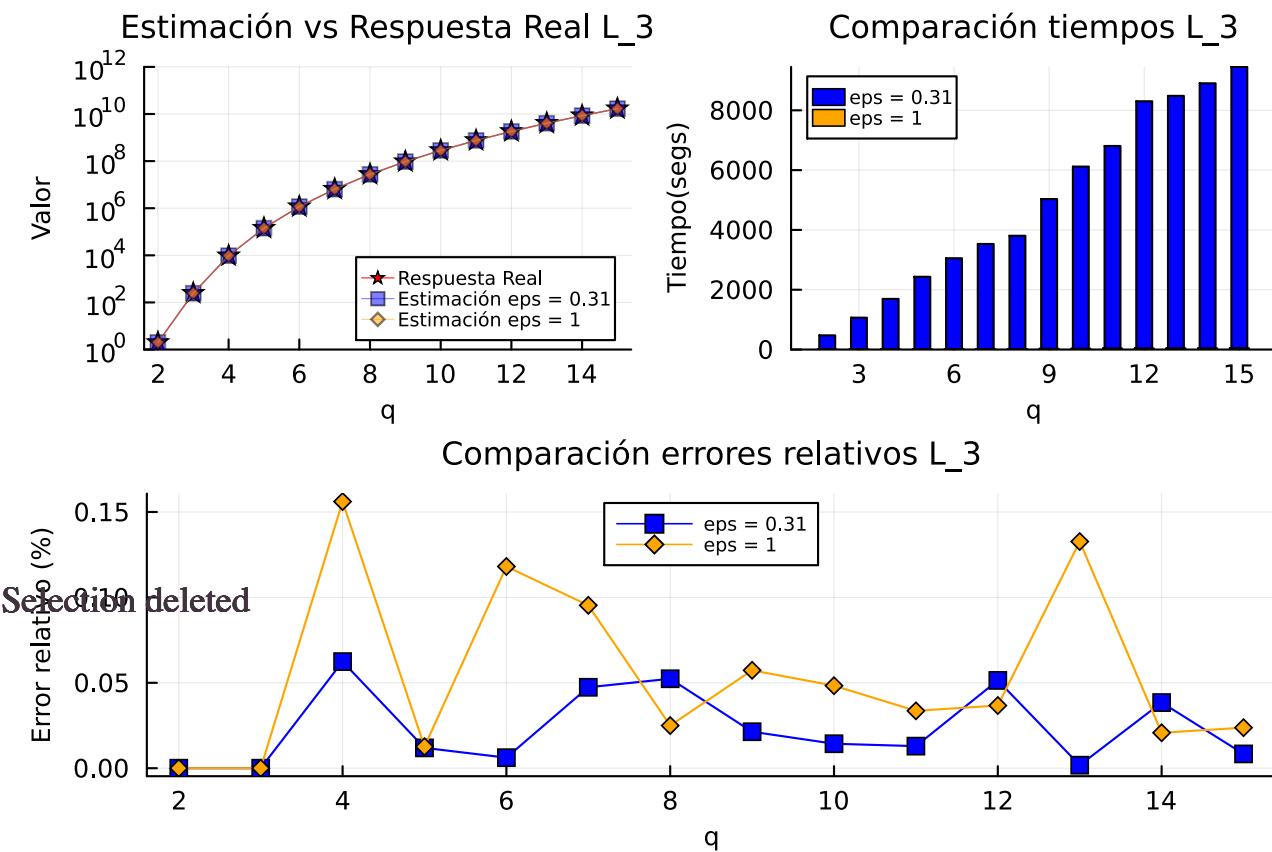
```
file_path_d = "../results/q-colorings/k=3/eps=1.csv"
```

```
1 dfd = CSV.File(file_path_d, delim=' ', header=true) |> DataFrame;
```

Pasos del Gibbs para $2 \leq q \leq 15$ ($\epsilon = 1$), respectivamente:

```
▶ [12, 15, 19, 22, 25, 29, 33, 36, 41, 45, 50, 55, 61, 67]
```

Visualización de los datos:



Nota: Los cálculos para $\epsilon = 0.31$ se hicieron con una versión no optimizada del código, de ahí los tiempos de ejecución.

Ahora, consideraremos resultados obtenidos con los siguientes parámetros:

$$\text{num_simulations} = \frac{n^3}{\epsilon^2}$$

$$\text{num_gibbs_steps} = n \left(\left| \frac{2 \log n + \log \epsilon^{-1} + \log 8}{\log \frac{q}{2d^2}} \right| + 1 \right)$$

El orden de complejidad total sigue siendo de $Cn^5 \log n$ sin embargo hemos reducido la constante C .

k=4

$$\epsilon = 0.1$$

num_simulations = 409600

```
file_path_e = "../results/q-colorings/k=4/eps=0.1.csv"
```

	q	Gibbs	Est	Res★	R_err	Mean_r	Time
1	2	41	2	2	0.0	0.68748	1.79
2	3	51	7782	7812	0.384	0.6996	6.13
3	4	60	6014108	6000732	0.2229	0.76061	7.51
4	5	69	827192957	828850160	0.1999	0.80464	14.3
5	6	78	38137020438	38128724910	0.02176	0.83584	15.4
6	7	88	860067118529	856858754052	0.3744	0.85877	19.1
7	8	98	11728066824679	11722360851992	0.04868	0.87597	22.3
8	9	109	111483647197418	111647093496192	0.1464	0.88949	27.3
9	10	120	806501388861500	807567269568570	0.132	0.90041	33.2
10	11	132	4659290565923617	4707230299664420	1.018	0.90904	39.0
11	12	145	23214815742732030	23062698161984052	0.6596	0.91718	45.5
12	13	160	97577491441800350	97963534144477872	0.3941	0.92313	49.2
13	14	176	369848796327724517	369313246327400102	0.145	0.92879	51.6
14	15	193	1251535692317397044	1258250118125770980	0.5336	0.93325	53.7

$$\epsilon = 0.5$$

num_simulations = 16384

```
file_path_f = "../results/q-colorings/k=4/eps=0.5.csv"
```

```
1 dff = CSV.File(file_path_f, delim=' ', header=true) |> DataFrame;
```

Pasos del Gibbs para $2 \leq q \leq 15$ ($\epsilon = 0.5$), respectivamente:

► [31, 40, 47, 55, 63, 71, 79, 88, 98, 108, 119, 131, 144, 159]

$$\epsilon = 1$$

`num_simulations = 4096`

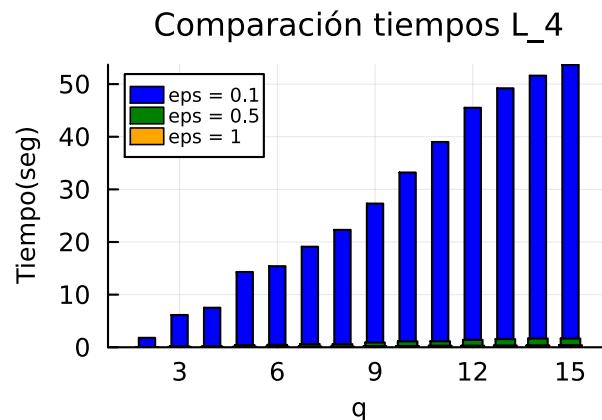
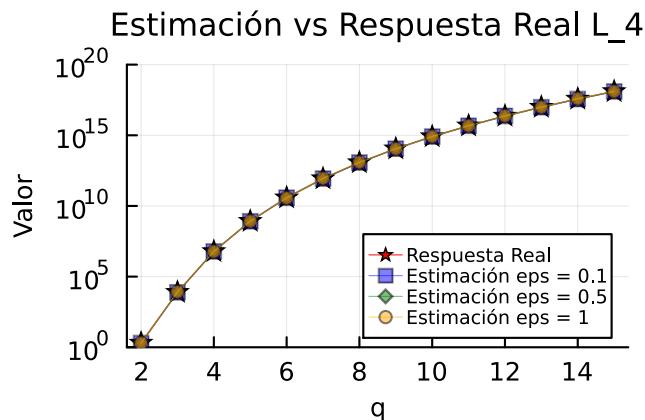
`file_path_g = ".../results/q-colorings/k=4/eps=1.csv"`

```
1 dfg = CSV.File(file_path_g, delim=' ', header=true) |> DataFrame;
```

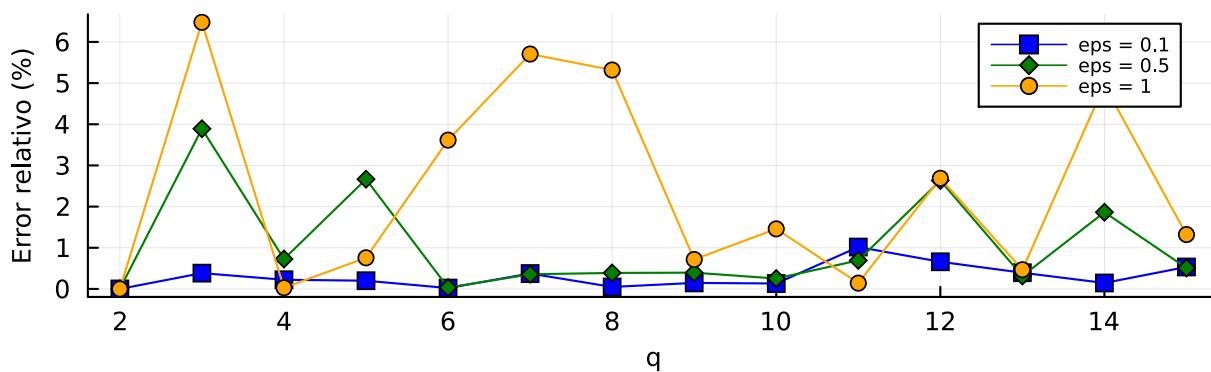
Pasos del Gibbs para $2 \leq q \leq 15$ ($\epsilon = 1$), respectivamente:

► [27, 35, 42, 49, 56, 64, 71, 80, 88, 98, 108, 119, 131, 145]

Visualización de los datos:



Comparación errores relativos L_4



Observación: Debido a que redujimos la constante en la complejidad, el error relativo en epsilon aumentó considerablemente.

k=5

$\epsilon = 0.1$

num_simulations = 1562500

`file_path_h = ".../results/q-colorings/k=5/eps=0.1.csv"`

<code>dfh =</code>	k	q	Gibbs	Est	Res★	R_err	Mean_r	Time
1	5	2	72	2.0	2.0	0.0	0.70002	18.3
2	5	3	89	580980.0	580990.0	0.001893	0.70267	62.7
3	5	4	105	2.0495e10	2.0443e10	0.2567	0.76134	103.0
4	5	5	120	5.0733e13	5.082e13	0.171	0.80499	148.0
5	5	6	136	2.201e16	2.1978e16	0.1442	0.83603	195.0
6	5	7	152	3.0278e18	3.0318e18	0.1325	0.8587	233.0
7	5	8	170	1.8956e20	1.8959e20	0.01348	0.87602	233.0
8	5	9	188	6.6682e21	6.6607e21	0.1118	0.88961	381.0
9	5	10	207	1.5133e23	1.5109e23	0.1585	0.90053	449.0
10	5	11	228	2.4292e24	2.4325e24	0.1369	0.90942	587.0
11	5	12	250	2.9786e25	2.9731e25	0.1851	0.91698	667.0
12	5	13	275	2.8931e26	2.8981e26	0.174	0.92325	679.0
13	5	14	302	2.3377e27	2.3387e27	0.04275	0.92873	739.0
14	5	15	332	1.6089e28	1.608e28	0.05557	0.93348	716.0

$\epsilon = 0.4$

num_simulations = 97656

`file_path_i = ".../results/q-colorings/k=5/eps=0.4.csv"`

```
1 dfl = CSV.File(file_path_i, delim=' ', header=true) |> DataFrame;
```

Pasos del Gibbs para $2 \leq q \leq 15$ ($\epsilon = 0.4$), respectivamente:

► [60, 74, 88, 102, 115, 130, 145, 160, 177, 195, 215, 236, 260, 286]

$$\epsilon = 0.8$$

`num_simulations = 24414`

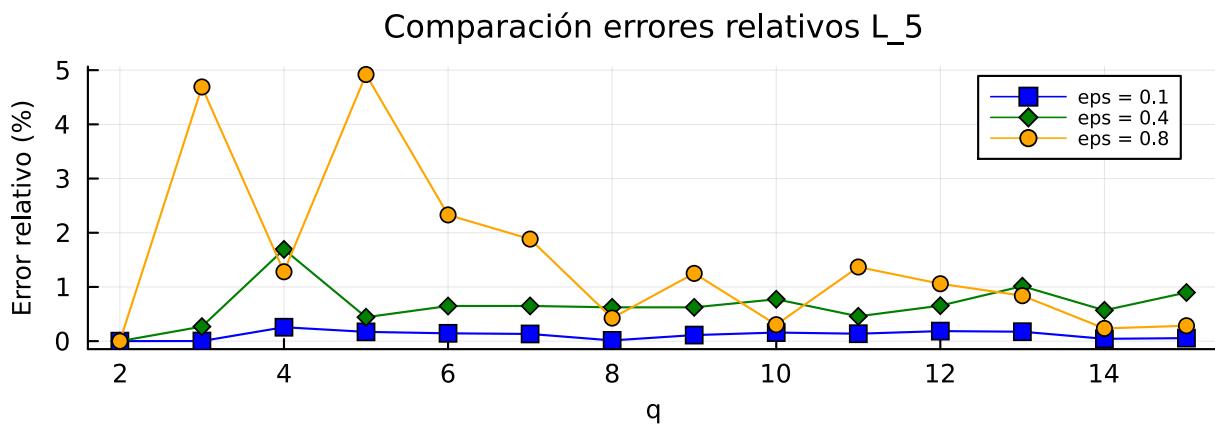
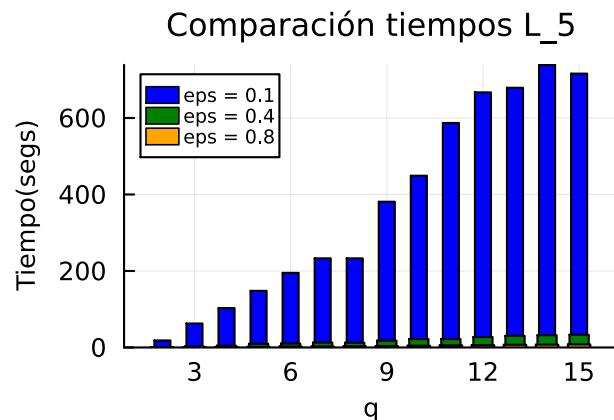
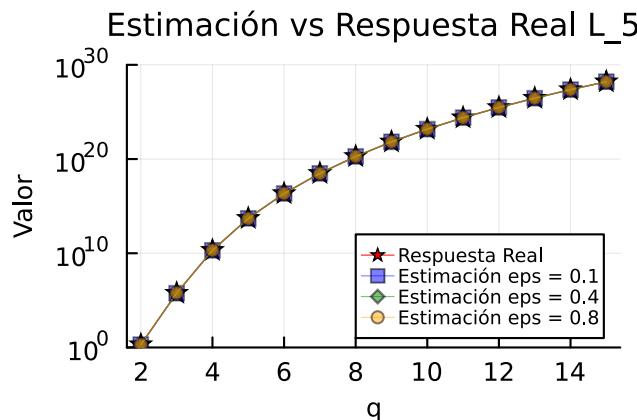
```
file_path_j = "../results/q-colorings/k=5/eps=0.8.csv"
```

```
1 dfj = CSV.File(file_path_j, delim=' ', header=true) |> DataFrame;
```

Pasos del Gibbs para $2 \leq q \leq 15$ ($\epsilon = 0.8$), respectivamente:

► [53, 67, 80, 92, 105, 118, 132, 147, 162, 179, 197, 217, 239, 263]

Visualización de los datos:



Observación: Aunque el teorema pide que $q > 2d = 8$, hasta el momento hemos conseguido muy

buenos resultados para $q \leq 8$.

k=6

$$\epsilon = 0.2$$

num_simulations = 1166400

`file_path_k = ".../results/q-colorings/k=6/eps=0.2.csv"`

<code>dfk =</code>	k	q	Gibbs	Est	Res★	R_err	Mean_r	Time
1	6	2	104	2.0	2.0	0.0	0.70825	33.5
2	6	3	129	1.0079e8	1.016e8	0.7985	0.7047	124.0
3	6	4	151	3.7982e14	3.8005e14	0.06206	0.76178	197.0
4	6	5	174	3.298e19	3.2921e19	0.1781	0.80526	262.0
5	6	6	197	2.2281e23	2.2348e23	0.2988	0.83607	392.0
6	6	7	221	2.8699e26	2.8636e26	0.2186	0.85883	495.0
7	6	8	245	1.1556e29	1.157e29	0.1257	0.87605	488.0
8	6	9	272	2.0123e31	2.0173e31	0.2495	0.88958	652.0
9	6	10	299	1.852e33	1.8597e33	0.4151	0.90045	798.0
10	6	11	329	1.0457e35	1.0409e35	0.4572	0.90954	823.0
11	6	12	362	3.8989e36	3.9027e36	0.09764	0.91694	901.0
12	6	13	397	1.0544e38	1.0532e38	0.1156	0.92332	913.0
13	6	14	436	2.1562e39	2.1601e39	0.1803	0.92872	951.0
14	6	15	479	3.5054e40	3.5111e40	0.1627	0.93345	996.0

$$\epsilon = 0.5$$

num_simulations = 186624

```
file_path_l = "../results/q-colorings/k=6/eps=0.5.csv"
```

```
1 dfl = CSV.File(file_path_l, delim=' ', header=true) |> DataFrame;
```

Pasos del Gibbs para $2 \leq q \leq 15$ ($\epsilon = 0.5$), respectivamente:

```
► [93, 115, 136, 156, 177, 199, 222, 246, 271, 299, 328, 361, 396, 436]
```

$$\epsilon = 0.8$$

num_simulations = 72900

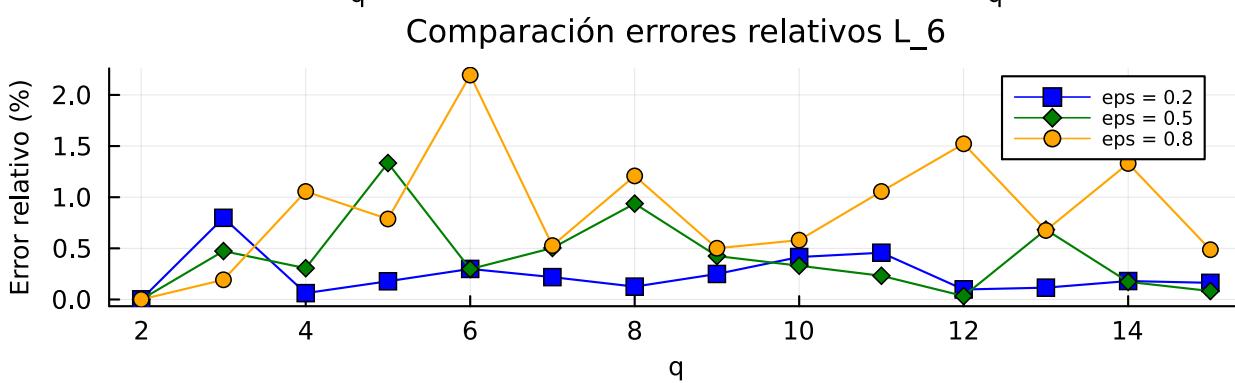
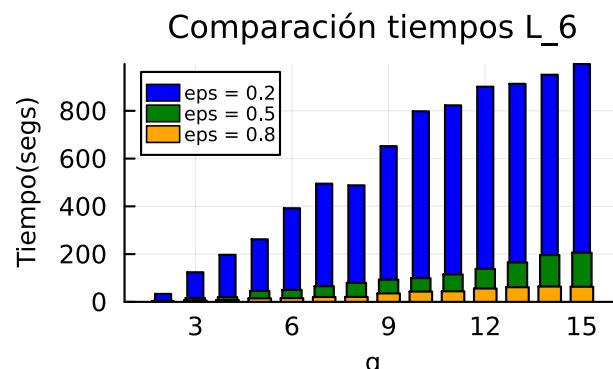
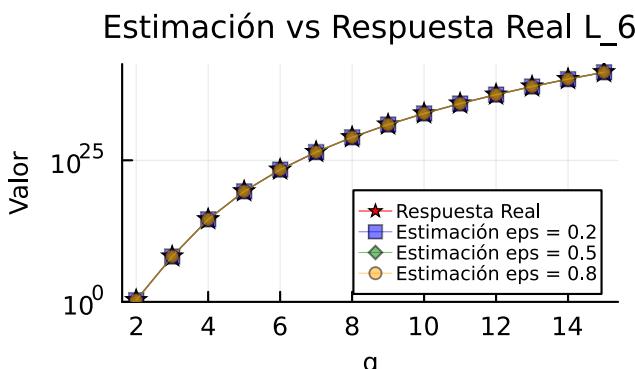
```
file_path_m = "../results/q-colorings/k=6/eps=0.8.csv"
```

```
1 dfm = CSV.File(file_path_m, delim=' ', header=true) |> DataFrame;
```

Pasos del Gibbs para $2 \leq q \leq 15$ ($\epsilon = 0.8$), respectivamente:

```
► [86, 108, 127, 147, 167, 188, 209, 232, 257, 283, 311, 342, 376, 413]
```

Visualización de los datos:



k=7

$$\epsilon = 0.4$$

num_simulations = 735306

file_path_n = ".../results/q-colorings/k=7/eps=0.4.csv"

dfn =	k	q	Gibbs	Est	Res★	R_err	Mean_r	Time
1	7	2	141	2.0	2.0	0.0	0.71428	48.1
2	7	3	174	4.2229e10	4.187e10	0.8564	0.70646	162.0
3	7	4	205	3.8126e19	3.8558e19	1.121	0.76205	249.0
4	7	5	235	2.2627e26	2.2531e26	0.4253	0.80543	362.0
5	7	6	266	4.0115e31	4.0087e31	0.06953	0.8362	475.0
6	7	7	298	7.198e35	7.2202e35	0.3077	0.85881	530.0
7	7	8	332	2.6625e39	2.6645e39	0.07696	0.87609	585.0
8	7	9	367	3.1118e42	3.1016e42	0.3281	0.88967	787.0
9	7	10	405	1.5018e45	1.5059e45	0.2725	0.9005	828.0
10	7	11	445	3.6892e47	3.6888e47	0.009851	0.90948	861.0
11	7	12	489	5.2165e49	5.2166e49	0.001308	0.91696	913.0
12	7	13	537	4.7007e51	4.7017e51	0.01991	0.9233	1010.0
13	7	14	589	2.9241e53	2.9099e53	0.4891	0.92881	1140.0
14	7	15	648	1.3129e55	1.31e55	0.2214	0.9335	1170.0

$$\epsilon = 0.6$$

num_simulations = 326802

file_path_o = ".../results/q-colorings/k=7/eps=0.6.csv"

```
1 dfo = CSV.File(file_path_o, delim=' ', header=true) |> DataFrame;
```

Pasos del Gibbs para $2 \leq q \leq 15$ ($\epsilon = 0.6$), respectivamente:

► [134, 165, 195, 224, 254, 285, 317, 351, 388, 427, 469, 515, 565, 621]

$$\epsilon = 0.8$$

`num_simulations = 183826`

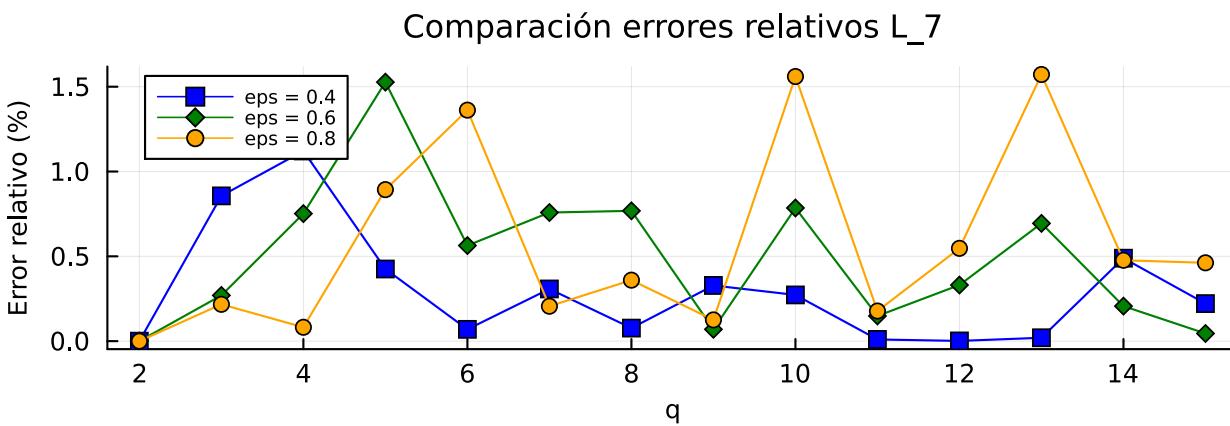
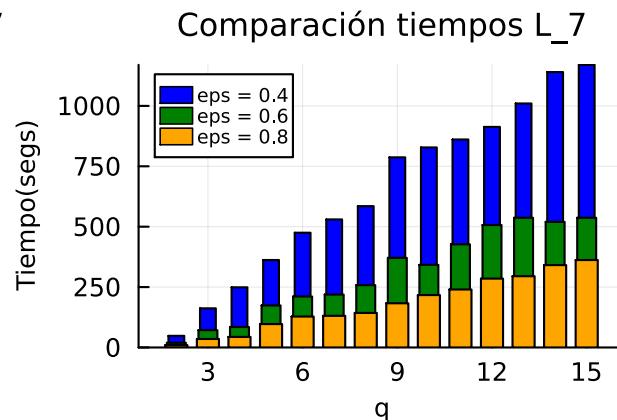
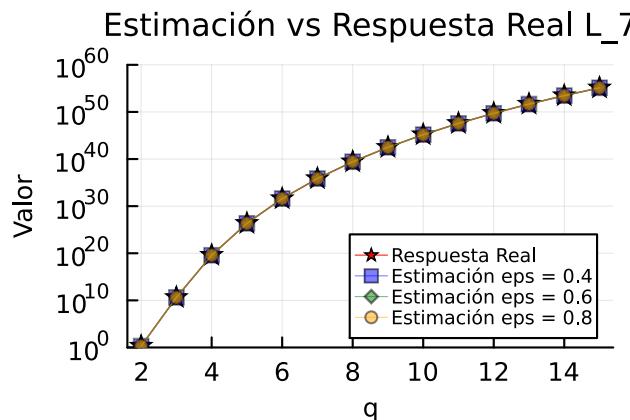
```
file_path_p = "../results/q-colorings/k=7/eps=0.8.csv"
```

```
1 dfp = CSV.File(file_path_p, delim=' ', header=true) |> DataFrame;
```

Pasos del Gibbs para $2 \leq q \leq 15$ ($\epsilon = 0.8$), respectivamente:

► [129, 159, 188, 217, 246, 276, 307, 340, 375, 413, 454, 499, 548, 603]

Visualización de los datos:



En este punto tenemos que decir que para $k \geq 8$ *Mathematica* ya no permite calcular el polinomio cromático debido al límite computacional sobre el plan de la prueba gratuita. Intentamos algunos recursos como calcular directamente el polinomio cromático con la librería NetworkX de Python o calcular el polinomio de Tutte con un programa que encontramos en lenguaje C, sin embargo al parecer $L_{k \geq 8}$ ya es un grafo lo suficientemente grande como para considerar estas opciones viables. Es por esto que para los siguientes resultados compararemos las estimaciones obtenidas con las respuestas que se muestran en esta página, que contiene resultados hasta $k = 13$ y $q = 10$.

Ahora, consideraremos resultados obtenidos con los siguientes parámetros:

$$\text{num_simulations} = \frac{n^3}{\epsilon^2}$$

$$\text{num_gibbs_steps} = n \left(\frac{\log n + \log \epsilon^{-1}}{\log \frac{q}{4d^2}} \right)$$

De nuevo, el orden de complejidad total sigue siendo de $Cn^5 \log n$ para alguna constante C .

k=8

$\epsilon = 0.4$

num_simulations = 1638400

```
file_path_q = "../results/q-colorings/k=8/eps=0.4.csv"
```

<code>dfq =</code>	<code>k</code>	<code>q</code>	<code>Gibbs</code>	<code>Est</code>	<code>Res★</code>	<code>R_err</code>	<code>Mean_r</code>	<code>Time</code>
1	8	2	93	2.0	2.0	0.0	0.71874	63.2
2	8	3	106	4.0899e13	4.0725e13	0.428	0.70766	154.0
3	8	4	117	2.114e25	2.13e25	0.7505	0.76235	184.0
4	8	5	127	1.6276e34	1.63e34	0.1485	0.80549	273.0
5	8	6	137	1.2589e41	1.27e41	0.8757	0.8362	597.0
6	8	7	146	4.8665e46	4.86e46	0.1335	0.85889	574.0
7	8	8	156	2.3225e51	2.32e51	0.1079	0.87614	668.0
8	8	9	165	2.4218e55	2.42e55	0.07288	0.88965	843.0
9	8	10	174	8.0334e58	8.02e58	0.1669	0.90055	875.0

$$\epsilon = 0.6$$

`num_simulations = 728177`

`file_path_r = "../results/q-colorings/k=8/eps=0.6.csv"`

```
1 dfr = CSV.File(file_path_r, delim=' ', header=true) |> DataFrame;
```

Pasos del Gibbs para $2 \leq q \leq 10$ ($\epsilon = 0.6$), respectivamente:

▶ [86, 97, 107, 117, 126, 135, 143, 152, 160]

$$\epsilon = 0.8$$

`num_simulations = 409600`

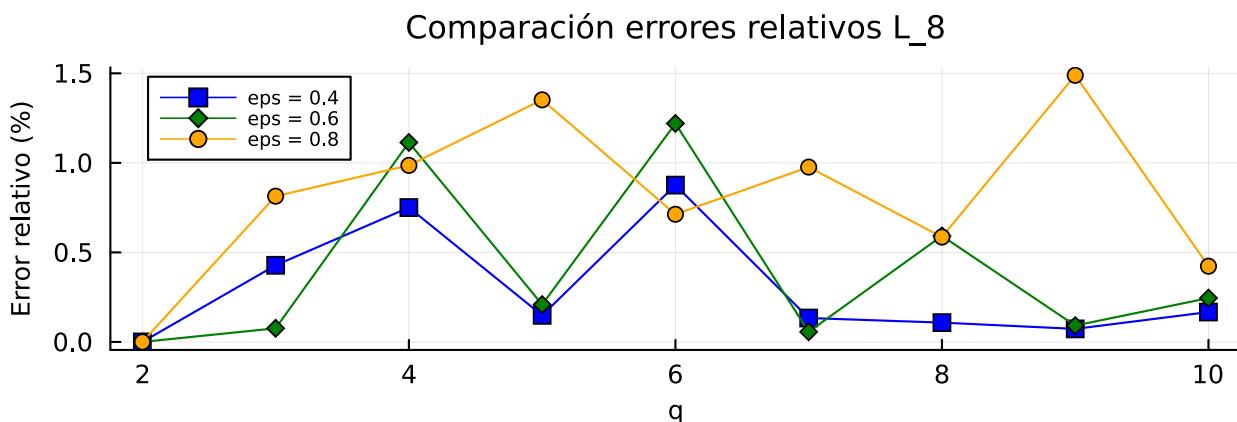
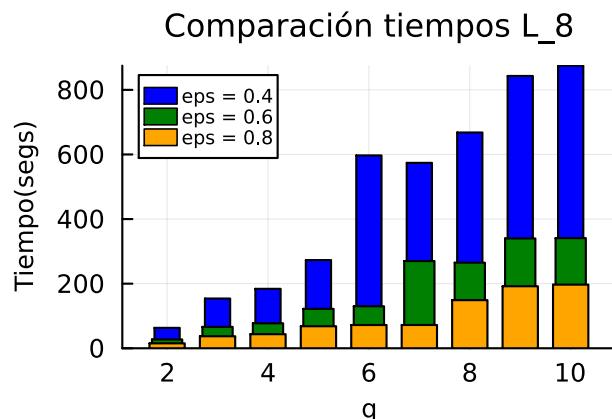
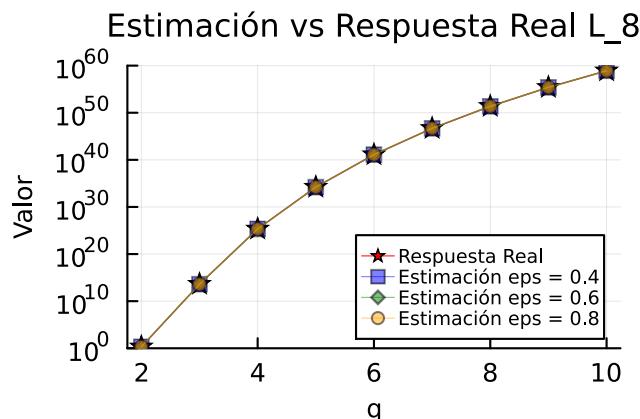
`file_path_s = "../results/q-colorings/k=8/eps=0.8.csv"`

```
1 dfs = CSV.File(file_path_s, delim=' ', header=true) |> DataFrame;
```

Pasos del Gibbs para $2 \leq q \leq 10$ ($\epsilon = 0.8$), respectivamente:

► [80, 91, 101, 110, 118, 126, 134, 142, 151]

Visualización de los datos:



$$k=9$$

$$\epsilon = 0.5$$

num_simulations = 2125764

```
file_path_t = ".../results/q-colorings/k=9/eps=0.5.csv"
```

dft =	k	q	Gibbs	Est	Res★	R_err	Mean_r	Time
1	9	2	118	2.0	2.0	0.0	0.72223	154
2	9	3	134	9.3976e16	9.3575e16	0.429	0.70866	358
3	9	4	148	6.4666e31	6.45e31	0.2581	0.76265	407
4	9	5	161	1.2439e43	1.24e43	0.3175	0.80559	664
5	9	6	174	7.1108e51	7.08e51	0.4348	0.83633	1372
6	9	7	186	8.7031e58	8.73e58	0.3079	0.85889	1423
7	9	8	198	7.6211e64	7.59e64	0.4092	0.87616	1286
8	9	9	210	9.6536e69	9.59e69	0.6629	0.8897	1565
9	9	10	221	2.8075e74	2.81e74	0.08885	0.90054	1521

$$\epsilon = 0.7$$

num_simulations = 1084573

file_path_u = "../results/q-colorings/k=9/eps=0.7.csv"

```
1 dfu = CSV.File(file_path_u, delim=' ', header=true) |> DataFrame;
```

Pasos del Gibbs para $2 \leq q \leq 10$ ($\epsilon = 0.7$), respectivamente:

► [111, 125, 138, 150, 162, 173, 185, 196, 207]

$$\epsilon = 0.9$$

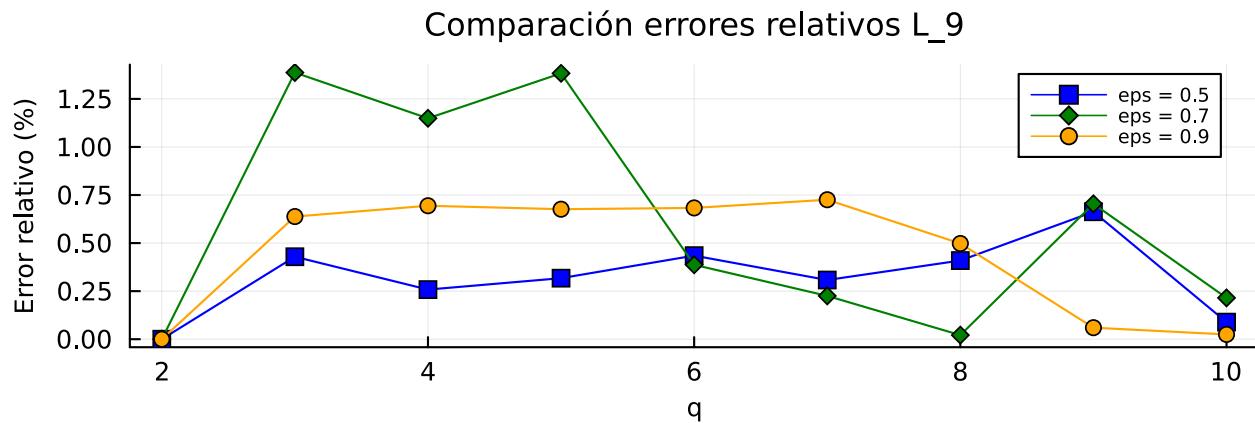
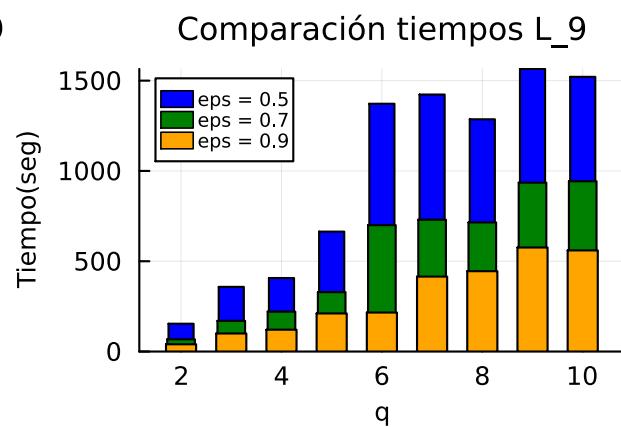
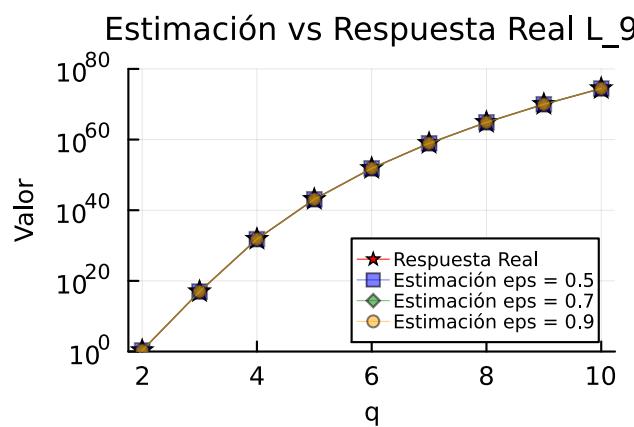
num_simulations = 656099

file_path_v = "../results/q-colorings/k=9/eps=0.9.csv"

```
1 dfv = CSV.File(file_path_v, delim=' ', header=true) |> DataFrame;
```

Pasos del Gibbs para $2 \leq q \leq 10$ ($\epsilon = 0.9$), respectivamente:

Visualización de los datos:



k=10

$\epsilon = 1$

`num_simulations = 1000000`

```
file_path_x = "../results/q-colorings/k=10/eps=1.csv"
```

<code>dfx =</code>	<code>k</code>	<code>q</code>	<code>Gibbs</code>	<code>Est</code>	<code>Res★</code>	<code>R_err</code>	<code>Mean_r</code>	<code>Time</code>
1	10	2	132	2.0	2.0	0.0	0.72498	113.62
2	10	3	150	5.0298e20	5.08e20	0.9875	0.70942	333.18
3	10	4	166	1.0693e39	1.06e39	0.8804	0.76283	384.59
4	10	5	180	9.9489e52	1.0e53	0.5112	0.80562	602.79
5	10	6	194	6.9235e63	6.97e63	0.6674	0.83631	647.64
6	10	7	208	4.1872e72	4.19e72	0.06574	0.85893	1305.8
7	10	8	221	9.3657e79	9.4e79	0.365	0.87613	1337.2
8	10	9	234	1.9197e86	1.93e86	0.5323	0.88965	1401.9
9	10	10	248	6.4896e91	6.48e91	0.1486	0.90056	1329.4

Para este ejemplo, fijaremos el número de simulaciones independiente del epsilon (solo afectará el número de pasos del Gibbs sampler).

$$\epsilon = 1$$

$$\text{num_simulations} = 100000$$

```
file_path_y = "../results/q-colorings/k=10/sims=1e5.csv"
```

```
1 dfy = CSV.File(file_path_y, delim=' ', header=true) |> DataFrame;
```

Pasos del Gibbs para $2 \leq q \leq 10$ ($\epsilon = 0.1$), respectivamente:

► [132, 150, 166, 180, 194, 208, 221, 234, 248]

Para este ejemplo, fijaremos el número de simulaciones independiente del epsilon (solo afectará el número de pasos del Gibbs sampler).

$$\epsilon = 1$$

$$\text{num_simulations} = 50000$$

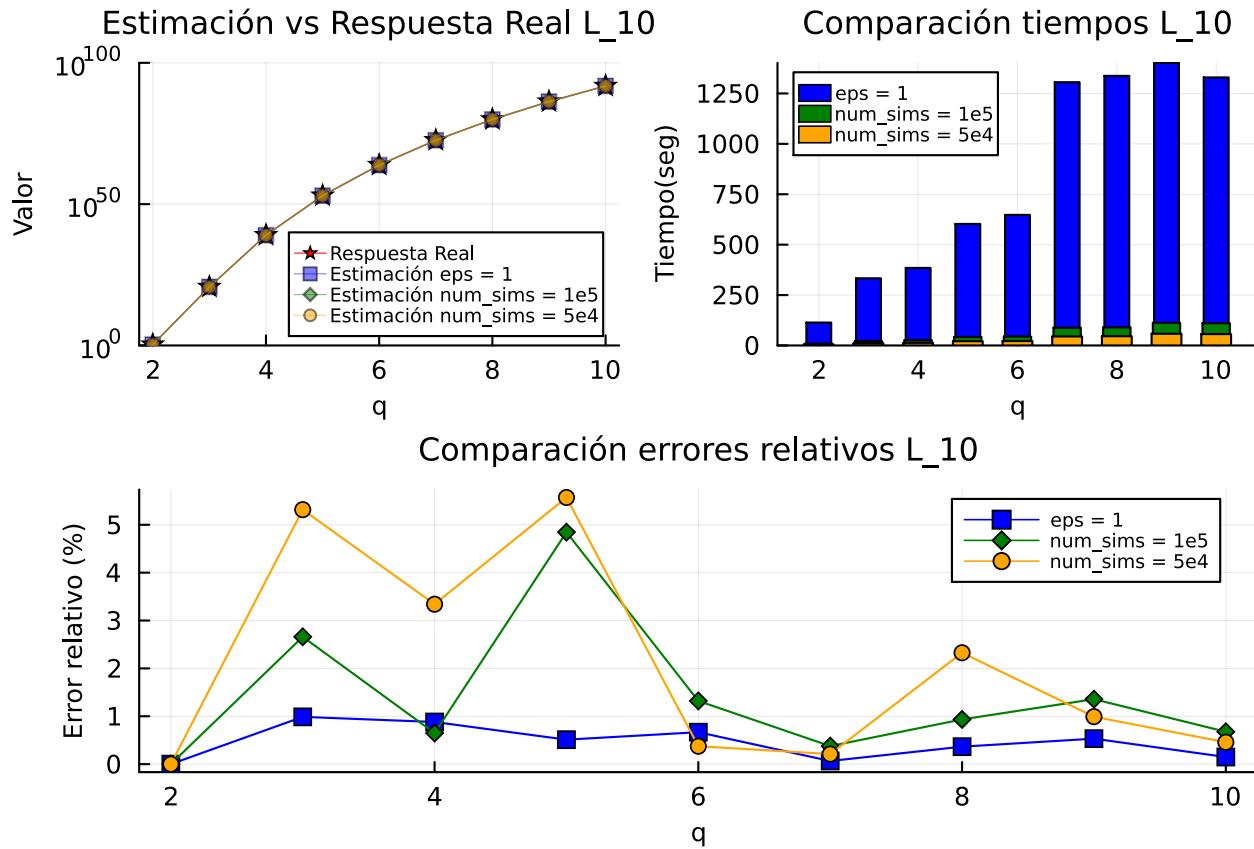
```
file_path_z = "../results/q-colorings/k=10/sims=5e4.csv"
```

```
1 dfz = CSV.File(file_path_z, delim=' ', header=true) |> DataFrame;
```

Pasos del Gibbs para $2 \leq q \leq 10$ ($\epsilon = 0.1$), respectivamente:

► [132, 150, 166, 180, 194, 208, 221, 234, 248]

Visualización de los datos:



k=11

$\epsilon = 1$

num_simulations = 1771561

```
file_path_aa = "../results/q-colorings/k=11/eps=1.csv"
```

<code>dfaa</code> =	<code>k</code>	<code>q</code>	<code>Gibbs</code>	<code>Est</code>	<code>Res★</code>	<code>R_err</code>	<code>Mean_r</code>	<code>Time</code>
1	11	2	167	2.0	2.0	0.0	0.72728	308.96
2	11	3	189	6.4933e24	6.53e24	0.5614	0.71015	699.73
3	11	4	209	9.5617e46	9.57e46	0.08631	0.76294	813.22
4	11	5	227	8.532e63	8.57e63	0.4437	0.8057	1255.5
5	11	6	245	1.2057e77	1.21e77	0.3525	0.83635	2578.0
6	11	7	262	5.3738e87	5.36e87	0.2575	0.85896	2390.3
7	11	8	279	4.3965e96	4.39e96	0.148	0.87617	2443.1
8	11	9	295	1.9818e104	1.97e104	0.5967	0.8897	2938.5
9	11	10	312	9.8055e110	9.84e110	0.351	0.90055	2837.4

Para este ejemplo, fijaremos el número de simulaciones independiente del epsilon (solo afectará el número de pasos del Gibbs sampler).

$$\epsilon = 1$$

$$\text{num_simulations} = 100000$$

```
file_path_ab = "../results/q-colorings/k=11/sims=1e5.csv"
```

```
1 dfab = CSV.File(file_path_ab, delim=' ', header=true) |> DataFrame;
```

Pasos del Gibbs para $2 \leq q \leq 10$ ($\epsilon = 0.1$), respectivamente:

► [167, 189, 209, 227, 245, 262, 279, 295, 312]

Para este ejemplo, fijaremos el número de simulaciones independiente del epsilon (solo afectará el número de pasos del Gibbs sampler).

$$\epsilon = 1$$

$$\text{num_simulations} = 50000$$

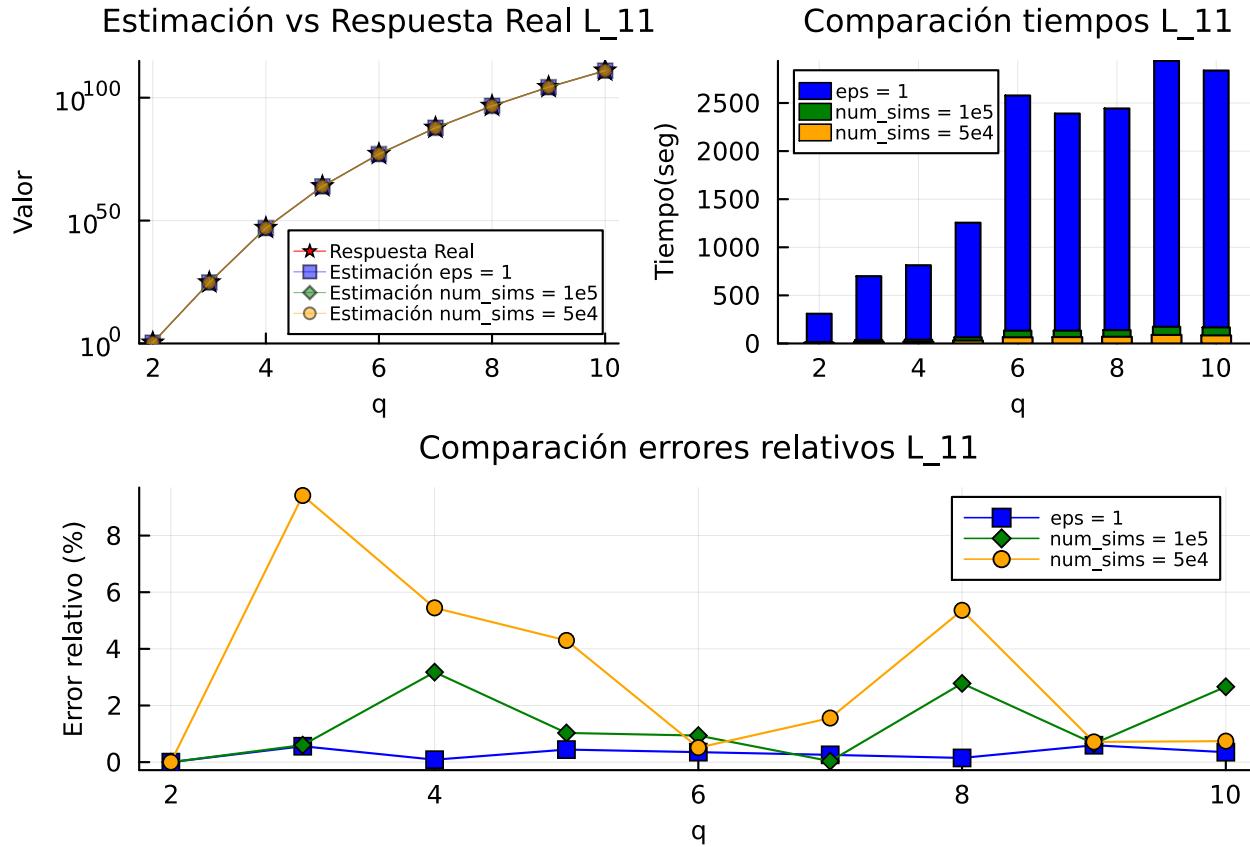
```
file_path_ac = "../results/q-colorings/k=11/sims=5e4.csv"
```

```
1 dfac = CSV.File(file_path_ac, delim=' ', header=true) |> DataFrame;
```

Pasos del Gibbs para $2 \leq q \leq 10$ ($\epsilon = 0.1$), respectivamente:

► [167, 189, 209, 227, 245, 262, 279, 295, 312]

Visualización de los datos:



Para este ejemplo, fijaremos el número de simulaciones independiente del epsilon (solo afectará el número de pasos del Gibbs sampler).

$$\epsilon = 1$$

num_simulations = 1000000

```
file_path_ad = "../results/q-colorings/k=12/sims=1e6.csv"
```

<code>dfad</code> =	<code>k</code>	<code>q</code>	<code>Gibbs</code>	<code>Est</code>	<code>Res★</code>	<code>R_err</code>	<code>Mean_r</code>	<code>Time</code>
1	12	2	206	2.0	2.0	0.0	0.72913	201.35
2	12	3	233	1.9638e29	1.98e29	0.819	0.71072	500.4
3	12	4	258	4.74e55	4.7e55	0.8521	0.76308	578.22
4	12	5	280	7.7525e75	7.72e75	0.4211	0.80578	897.39
5	12	6	302	3.6766e91	3.71e91	0.901	0.83637	1825.3
6	12	7	323	1.8607e104	1.83e104	1.678	0.85901	1836.8
7	12	8	344	7.7866e114	7.73e114	0.7319	0.87619	1893.7
8	12	9	364	1.016e124	1.02e124	0.3934	0.88967	2660.8
9	12	10	385	9.7639e131	9.82e131	0.5711	0.90055	2337.3

Para este ejemplo, fijaremos el número de simulaciones independiente del epsilon (solo afectará el número de pasos del Gibbs sampler).

$$\epsilon = 1$$

$$\text{num_simulations} = 100000$$

```
file_path_ae = "../results/q-colorings/k=12/sims=1e5.csv"
```

```
1 dfae = CSV.File(file_path_ae, delim=' ', header=true) |> DataFrame;
```

Pasos del Gibbs para $2 \leq q \leq 10$ ($\epsilon = 1$), respectivamente:

► [206, 233, 258, 280, 302, 323, 344, 364, 385]

Para este ejemplo, fijaremos el número de simulaciones independiente del epsilon (solo afectará el número de pasos del Gibbs sampler).

$$\epsilon = 1$$

$$\text{num_simulations} = 50000$$

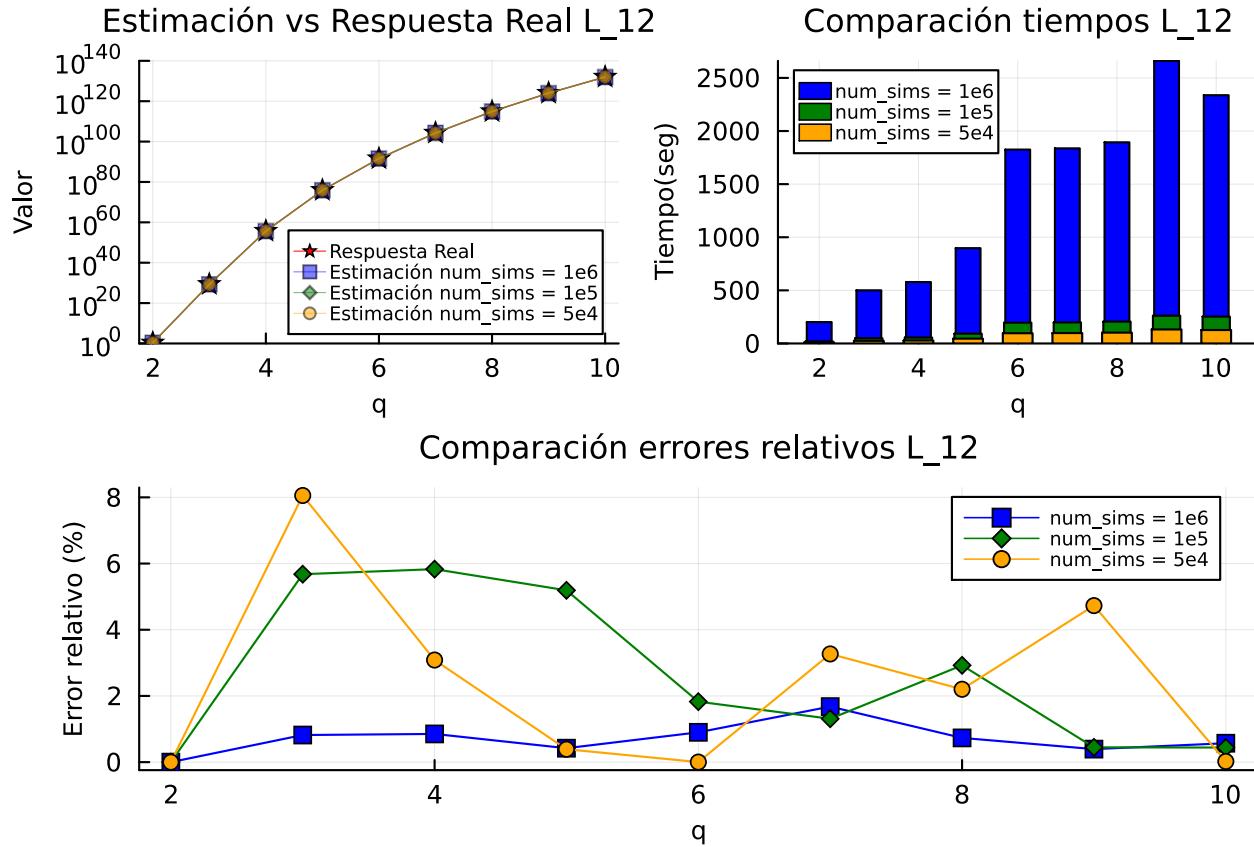
```
file_path_af = "../results/q-colorings/k=12/sims=5e4.csv"
```

```
1 dfaf = CSV.File(file_path_af, delim=' ', header=true) |> DataFrame;
```

Pasos del Gibbs para $2 \leq q \leq 10$ ($\epsilon = 1$), respectivamente:

► [206, 233, 258, 280, 302, 323, 344, 364, 385]

Visualización de los datos:



k=13

Para este ejemplo, fijaremos el número de simulaciones independiente del epsilon (solo afectará el número de pasos del Gibbs sampler).

$$\epsilon = 1$$

num_simulations = 1000000

```
file_path_ag = "../results/q-colorings/k=13/sims=1e6.csv"
```

<code>dfag</code> =	<code>k</code>	<code>q</code>	<code>Gibbs</code>	<code>Est</code>	<code>Res★</code>	<code>R_err</code>	<code>Mean_r</code>	<code>Time</code>
1	13	2	250	2.0	2.0	0.0	0.73079	386.77
2	13	3	283	1.4244e34	1.43e34	0.4154	0.71125	559.24
3	13	4	312	1.2494e65	1.26e65	0.8441	0.76314	821.06
4	13	5	340	7.3453e88	7.35e88	0.06343	0.80581	2084.9
5	13	6	366	2.0039e107	2.01e107	0.3059	0.83641	2207.0
6	13	7	391	1.6541e122	1.67e122	0.95	0.85895	2516.5
7	13	8	416	5.149e134	5.14e134	0.1749	0.87618	2579.9
8	13	9	441	2.6889e145	2.69e145	0.04008	0.88969	2673.2
9	13	10	467	6.4943e154	6.45e154	0.6874	0.90059	2817.4

Para este ejemplo, fijaremos el número de simulaciones independiente del epsilon (solo afectará el número de pasos del Gibbs sampler).

$$\epsilon = 1$$

$$\text{num_simulations} = 100000$$

```
file_path_ah = "../results/q-colorings/k=13/sims=1e5.csv"
```

```
1 dfah = CSV.File(file_path_ah, delim=' ', header=true) |> DataFrame;
```

Pasos del Gibbs para $2 \leq q \leq 10$ ($\epsilon = 1$), respectivamente:

► [250, 283, 312, 340, 366, 391, 416, 441, 467]

Para este ejemplo, fijaremos el número de simulaciones independiente del epsilon (solo afectará el número de pasos del Gibbs sampler).

$$\epsilon = 1$$

$$\text{num_simulations} = 50000$$

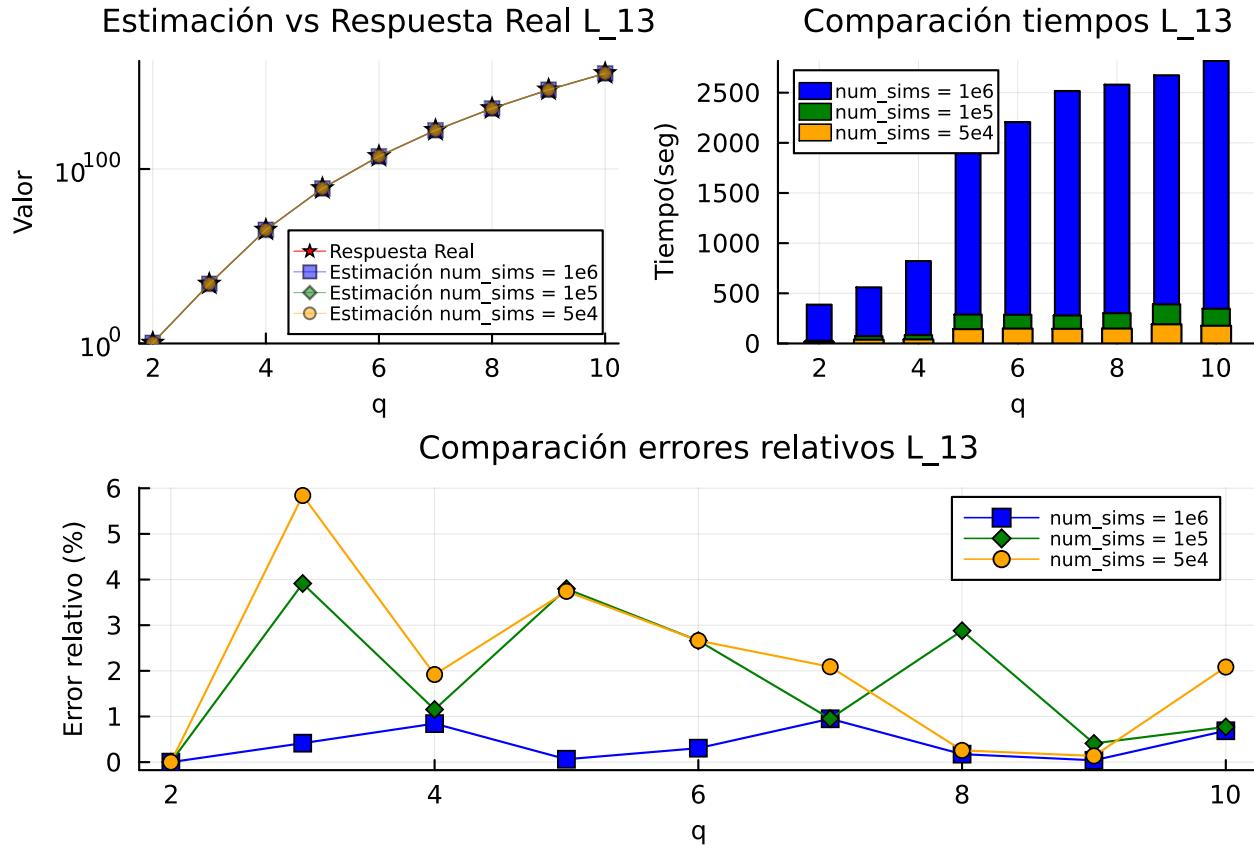
```
file_path_ai = "../results/q-colorings/k=13/sims=5e4.csv"
```

```
1 dfa1 = CSV.File(file_path_ai, delim=' ', header=true) |> DataFrame;
```

Pasos del Gibbs para $2 \leq q \leq 10$ ($\epsilon = 1$), respectivamente:

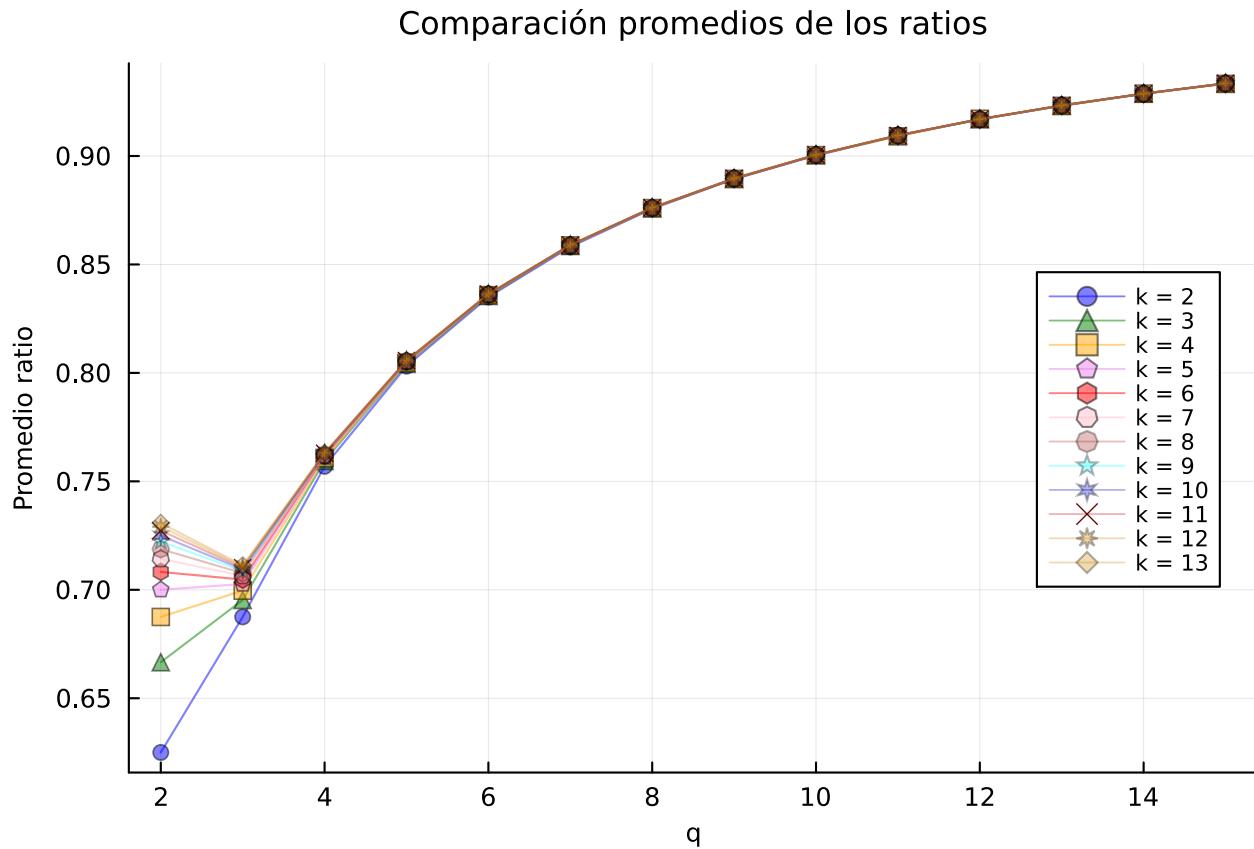
► [250, 283, 312, 340, 366, 391, 416, 441, 467]

Visualización de los datos:



Nota: No consideramos $k > 13$ pues no encontramos resultados exactos sobre estos valores para realizar comparaciones, y como mencionamos anteriormente calcular dichos polinomios cromáticos ya no es una opción muy viable. Además, los tiempos de ejecución de los programas ya estaban siendo altos incluso para $\epsilon \approx 1$ (Aunque tiempos aún bajos realizando un número de simulaciones fijo de hasta $1e5$, obteniendo errores relativos en su mayoría por debajo de 4% -varios por debajo del 2%-).

Promedio de los ratios:



Según estos datos podríamos decir que el promedio de los ratios es independiente de k (al menos para $5 \leq q \leq 15$ y $2 \leq k \leq 13$), ¿Se mantiene esto para cualquier $k > 13$ y cualquier $q > 15$?

Teniendo esto en cuenta, a priori podemos realizar las siguientes estimaciones para el número de q -coloraciones ($5 \leq q \leq 15$) de un lattice $k \times k$ ($k > 13$) con los valores calculados del promedio de los ratios como:

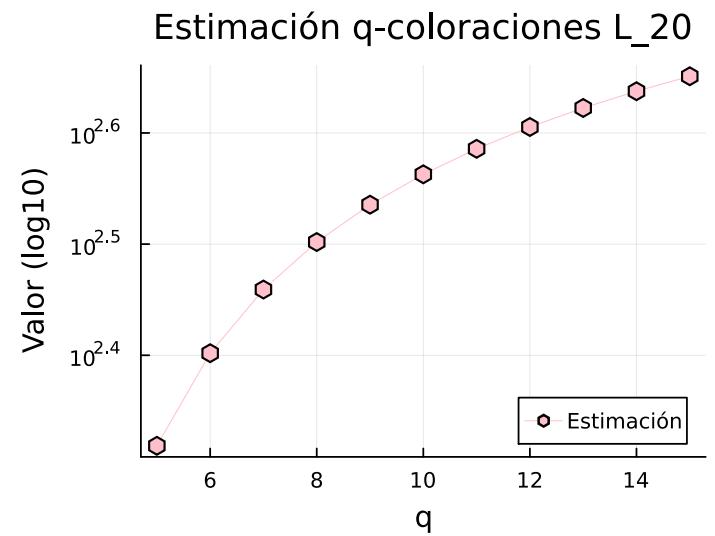
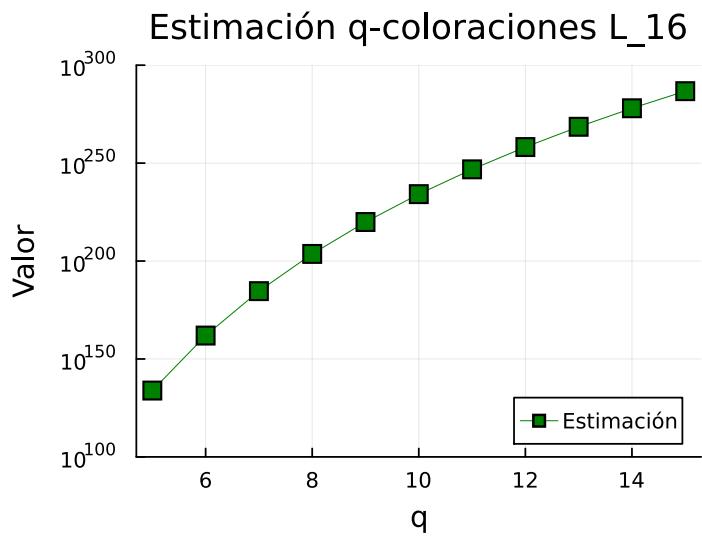
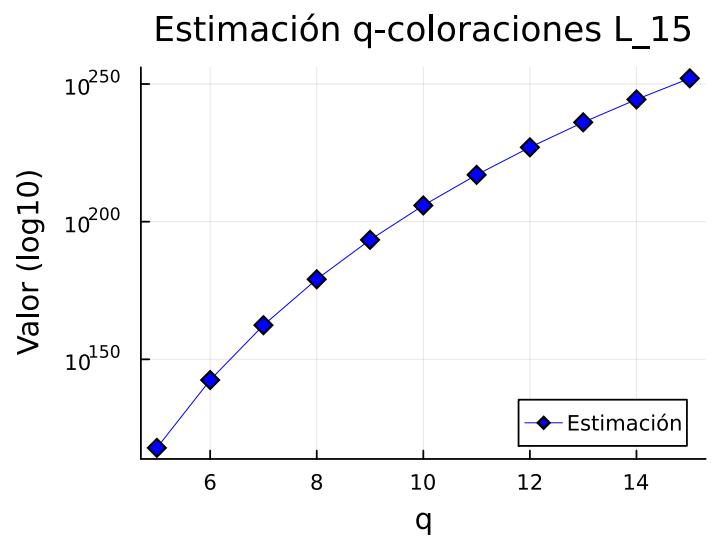
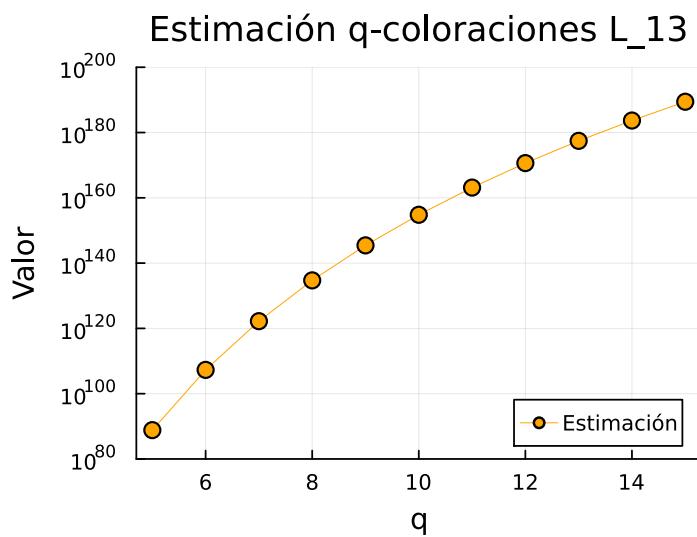
$$q^n * (r_q)^m$$

donde r_q es algún promedio de los ratios para el color q en algún k calculado anteriormente. Para la siguiente muestra tomamos los promedios anteriormente calculados para $k = 9$.

```
file_path_est = "../results/q-colorings/k=14..20/est.csv"
```

k\q	5	6	7	8	9	10
1	13	6.81584e+88	1.94714e+107	1.62212e+122	5.10782e+134	2.69764e+145
2	14	6.66256e+102	1.83282e+124	3.91276e+141	1.27645e+156	3.59914e+168
3	15	6.85739e+117	3.03847e+142	2.51668e+162	1.20305e+179	2.4371e+193
4	16	7.43144e+133	8.87165e+161	4.31639e+184	4.27641e+203	8.37541e+219
5	17	8.47976e+150	4.56212e+182	1.97405e+208	5.73308e+229	1.46083e+248
6	18	1.0188e+169	4.13183e+204	2.40737e+233	2.89875e+257	1.29315e+278
7	19	1.28882e+188	6.5907e+227	7.82842e+259	5.52772e+286	5.80981e+309
8	20	1.71669e+208	1.85154e+252	6.78815e+287	3.97554e+317	1.32475e+343

Algunas gráficas de los datos obtenidos:



Conclusiones:

- El método aproximado demostró ser efectivo y eficiente, comparado con métodos exactos que pueden ser computacionalmente inviables para grafos grandes.
- Los errores relativos fueron generalmente bajos, lo que sugiere que el método de aproximación, a pesar de su naturaleza estocástica, proporciona estimaciones razonablemente precisas.
- La implementación logró resultados útiles en tiempos razonables incluso para configuraciones de grafo más grandes, destacando la utilidad práctica del algoritmo en el problema de contar las q -coloraciones de un grafo reticular.
- El algoritmo proporciona un control razonable sobre el error, lo que permite confiar en la precisión de las estimaciones para la toma de decisiones.
- La variación en el número de simulaciones y los pasos del sampler muestra que el algoritmo es sensible a estos parámetros, lo que subraya la importancia de una configuración cuidadosa para maximizar la eficiencia y la efectividad.
- Existe espacio para optimizar aún más el algoritmo ajustando dinámicamente los parámetros en respuesta a las características del grafo y los resultados intermedios de las simulaciones, incluso se podría hacer uso de computación paralela para implementar varios hilos al momento de realizar las estimaciones de los ratios (esta probablemente sería una optimización clave y como anotación, un compañero nuestro ajeno a la clase lo está intentando hacer).

Segundo Punto

Obtenga valores aproximados para el número de configuraciones factibles en el modelo "Hard Core" para lattices $k \times k$ con $3 \leq k \leq 20$.

Reporte de manera similar a lo hecho en el item (a) del ejercicio anterior.

1 Enter cell code...

