# Resolução Heurística do Problema L(p,q)-Coloring

Lucas França Neves
Matheus Leal Costa
Gabriel Giacomo Paes
Álvaro Davi Carneiro dos Santos
João Victor Pereira

DCC059 - Teoria dos Grafos
Universidade Federal de Juiz de Fora

22 de janeiro de 2026

# Sumário

**1 Introdução** **3**
1.1 Definição do Problema . . . . . 3

**2 Algoritmos Implementados** **3**
2.1 Algoritmo Guloso . . . . . 3
2.2 Algoritmo Guloso Randomizado . . . . . 4
2.3 Algoritmo Guloso Randomizado Reativo . . . . . 4

**3 Metodologia Experimental** **5**
3.1 Instâncias Utilizadas . . . . . 5
3.2 Configuração dos Experimentos . . . . . 5

**4 Resultados** **6**
4.1 Resultados para Instância exemplo.col (10 vértices, 15 arestas) . . . . . 6
4.2 Resultados para Instância r250.5.col (250 vértices, 14849 arestas) . . . . . 6
4.3 Análise Comparativa . . . . . 6

**5 Análise dos Resultados** **6**

**6 Conclusão** **7**

# 1 Introdução

O problema L(p,q)-coloring é uma generalização do problema clássico de coloração de grafos, com aplicações importantes em alocação de frequências em redes de comunicação sem fio. Neste contexto, estações próximas devem usar frequências suficientemente distantes para evitar interferências.

## 1.1 Definição do Problema

Dado um grafo simples não direcionado $G = (V, E)$ e dois inteiros não negativos $p$ e $q$, o problema L(p,q)-coloring consiste em encontrar uma coloração $f : V \to \mathbb{Z}^+$ dos vértices tal que:

- Se $u$ e $v$ são vértices adjacentes (distância 1), então $|f(u) - f(v)| \geq p$
- Se $u$ e $v$ estão a distância 2, então $|f(u) - f(v)| \geq q$

O objetivo é minimizar a maior cor utilizada na coloração.

# 2 Algoritmos Implementados

## 2.1 Algoritmo Guloso

O algoritmo guloso implementado utiliza uma heurística construtiva aprimorada e determinística. A estratégia consiste em ordenar os vértices considerando tanto o grau direto quanto o número de vizinhos a distância 2, e então colorir cada vértice sequencialmente com a menor cor válida possível.

A ordenação aprimorada é crucial para o problema L(p,q)-coloring, pois vértices com maior grau combinado (direto + distância 2) tendem a ter mais restrições. Ao colori-los primeiro, reduz-se significativamente a chance de conflitos nas etapas finais. Para cada vértice, o algoritmo busca a menor cor (iniciando em 0) que satisfaça ambas as restrições: diferença $\geq p$ com vizinhos diretos e diferença $\geq q$ com vizinhos a distância 2.

Esta heurística mostrou-se superior à ordenação simples por grau, produzindo soluções de melhor qualidade em grafos densos.

**Algorithm 1** Algoritmo Guloso para L(p,q)-Coloring

$V \leftarrow$ vértices ordenados por $(grau + |N_2|)$ decrescente ▷ $N_2$ = vizinhos a distância 2
$coloring \leftarrow$ vetor de tamanho $|V|$ inicializado com $-1$
$maxColor \leftarrow 0$
**for** cada vértice $v \in V$ **do**
    $cor \leftarrow 0$
    **while** $\neg$ isValidColor$(v, cor, coloring)$ **do**
        $cor \leftarrow cor + 1$
    **end while**
    $coloring[v] \leftarrow cor$
    $maxColor \leftarrow \max(maxColor, cor)$
**end for**
**return** $(coloring, maxColor)$

## 2.2 Algoritmo Guloso Randomizado

O algoritmo guloso randomizado é baseado na técnica GRASP (Greedy Randomized Adaptive Search Procedure). Em vez de sempre escolher o vértice com menor custo, ele introduz aleatoriedade através de uma Lista Restrita de Candidatos (RCL).

O parâmetro $\alpha \in [0, 1]$ controla o grau de aleatoriedade: com $\alpha = 0$, o algoritmo é puramente guloso (escolhe sempre o melhor candidato), enquanto $\alpha = 1$ torna a escolha completamente aleatória. O custo de cada vértice não colorido combina a menor cor válida necessária com o **grau de saturação** (número de vizinhos já coloridos), priorizando vértices mais restritos. Especificamente, o custo é definido como:

$$cost(v) = minColor(v) \times 100 - saturationDegree(v)$$

A RCL contém todos os vértices cujo custo está dentro de um limiar $threshold = minCost + \alpha \cdot (maxCost - minCost)$.

O algoritmo executa múltiplas iterações (geralmente 30-50) e, ao final, aplica uma **busca local** para refinar a melhor solução encontrada, tentando recolorir vértices com cores altas em cores menores.

**Algorithm 2** Algoritmo Guloso Randomizado

$bestSol \leftarrow \emptyset$
**for** $iter = 1$ até $iterations$ **do**
    $coloring \leftarrow$ vetor inicializado com $-1$
    $uncolored \leftarrow$ todos os vértices
    **while** $uncolored \neq \emptyset$ **do**
        **for** cada $v \in uncolored$ **do**
            $minColor \leftarrow$ findSmallestValidColor$(v, coloring)$
            $satDegree \leftarrow$ número de vizinhos coloridos de $v$
            $cost[v] \leftarrow minColor \times 100 - satDegree$
        **end for**
        $minCost \leftarrow \min(cost)$, $maxCost \leftarrow \max(cost)$
        $threshold \leftarrow minCost + \alpha \cdot (maxCost - minCost)$
        $RCL \leftarrow \{v \in uncolored : cost[v] \leq threshold\}$
        $v^* \leftarrow$ escolher aleatoriamente de $RCL$
        $coloring[v^*] \leftarrow$ findSmallestValidColor$(v^*, coloring)$
        $uncolored \leftarrow uncolored \setminus \{v^*\}$
    **end while**
    $bestSol \leftarrow$ atualizar se solução atual for melhor
**end for**
$bestSol \leftarrow$ localSearch$(bestSol)$ ▷ Refinamento
**return** $bestSol$

## 2.3 Algoritmo Guloso Randomizado Reativo

O algoritmo reativo é uma extensão do GRASP que adapta automaticamente o parâmetro $\alpha$ durante a execução. Em vez de usar um único valor fixo de $\alpha$, ele trabalha com um conjunto de valores candidatos cuidadosamente escolhidos e ajusta dinamicamente as probabilidades de seleção de cada $\alpha$ baseado no desempenho observado.

Nossa implementação utiliza valores de $\alpha$ mais agressivos: $\{0.02, 0.05, 0.10, 0.15\}$, que favorecem escolhas mais gulosas (menor aleatoriedade), resultando em soluções de melhor qualidade para grafos densos. Valores menores de $\alpha$ concentram a busca em candidatos promissores, enquanto ainda mantendo diversificação suficiente.

O algoritmo divide as iterações em blocos (geralmente de tamanho 40-50). Ao final de cada bloco, ele avalia a qualidade média das soluções obtidas com cada $\alpha$ e atualiza as probabilidades de forma proporcional: valores de $\alpha$ que geraram soluções de melhor qualidade recebem probabilidades maiores de serem selecionados no próximo bloco. A qualidade é medida como $q = \frac{1}{1+maxColor}$, de modo que soluções com menor cor máxima têm maior qualidade.

Após todas as iterações, aplica-se uma **busca local mais extensa** (até 100 iterações) para refinar a melhor solução encontrada. A implementação é modularizada, com métodos auxiliares para construção de soluções (`buildSolution`) e atualização de probabilidades (`updateProbabilities`), facilitando manutenção e extensões futuras.

Este mecanismo de aprendizado permite que o algoritmo se adapte às características específicas de cada instância, privilegiando os valores de $\alpha$ mais eficazes.

**Algorithm 3** Algoritmo Guloso Randomizado Reativo

$alphas \leftarrow \{0.02, 0.05, 0.10, 0.15\}$
$probabilities \leftarrow$ distribuição uniforme sobre $alphas$
$bestSol \leftarrow \emptyset$, $blockQuality \leftarrow 0$
**for** $iter = 1$ até $iterations$ **do**
    $\alpha \leftarrow$ selecionar de $alphas$ com probabilidades
    $sol \leftarrow$ buildSolution($\alpha$) ▷ Método modularizado
    $quality \leftarrow \frac{1}{1+sol.maxColor}$
    $blockQuality[\alpha] \leftarrow blockQuality[\alpha] + quality$
    $bestSol \leftarrow$ atualizar se $sol$ for melhor
    **if** $iter \mod blockSize = 0$ **then**
        updateProbabilities($probabilities, blockQuality, blockUsage$)
        $blockQuality \leftarrow$ resetar para próximo bloco
    **end if**
**end for**
$bestSol \leftarrow$ localSearch($bestSol, 100$) ▷ Refinamento extenso
**return** $bestSol$

# 3 Metodologia Experimental

## 3.1 Instâncias Utilizadas

As instâncias utilizadas foram obtidas do repositório DIMACS e de benchmarks clássicos da literatura:

## 3.2 Configuração dos Experimentos

- Cada algoritmo foi executado 10 vezes para cada instância (com sementes 1-10)
- Valores de $p = 2$ e $q = 1$

Tabela 1: Instâncias de teste

| Instância | Vértices | Arestas | Descrição |
|---|---|---|---|
| exemplo.col | 10 | 15 | Instância pequena de teste |
| r250.5.col | 250 | 14849 | Grafo aleatório denso |

- Algoritmo randomizado: $\alpha = 0.3$, 50 iterações
- Algoritmo reativo: $\alpha \in \{0.02, 0.05, 0.10, 0.15\}$, 100 iterações com blocos de 30
- Ambiente: macOS, compilador clang++ com otimização -O2

# 4 Resultados

## 4.1 Resultados para Instância exemplo.col (10 vértices, 15 arestas)

Tabela 2: Resultados para exemplo.col ($p = 2$, $q = 1$)

| Algoritmo | Melhor | Média | Pior | Desvio | Tempo Médio (s) |
|---|---|---|---|---|---|
| Guloso | 6 | 6.00 | 6 | 0.00% | 0.000007 |
| Randomizado ($\alpha = 0.3$) | 6 | 6.00 | 6 | 0.00% | 0.000399 |
| Reativo | 6 | 6.00 | 6 | 0.00% | 0.000394 |

Para a instância pequena, todos os algoritmos encontraram a mesma solução ótima (cor máxima = 6), demonstrando que o problema é trivial para grafos pequenos.

## 4.2 Resultados para Instância r250.5.col (250 vértices, 14849 arestas)

Tabela 3: Resultados para r250.5.col ($p = 2$, $q = 1$)

| Algoritmo | Melhor | Média | Pior | Desvio[1] | Tempo Médio (s) |
|---|---|---|---|---|---|
| Guloso | 210 | 210.00 | 210 | 3.96% | 0.003 |
| Randomizado ($\alpha = 0.3$) | 221 | 223.30 | 225 | 10.54% | 9.53 |
| Reativo | **202** | 203.00 | 205 | **0.00%** | 20.26 |

## 4.3 Análise Comparativa

# 5 Análise dos Resultados

Os experimentos revelaram resultados interessantes e alguns contraintuitivos:

[1]Desvio percentual em relação à melhor solução encontrada (202 pelo algoritmo reativo).

Tabela 4: Comparação de desempenho entre algoritmos (r250.5.col)

| **Algoritmo** | **Qualidade** | **Velocidade** | **Consistência** |
|---|---|---|---|
| Guloso | Boa (210) | Muito Rápido | Alta (determinístico) |
| Randomizado | Inferior (221-225) | Médio | Baixa (variância alta) |
| Reativo | **Melhor (202-205)** | Lento | Alta (variância baixa) |

**Algoritmo Guloso:** Surpreendentemente, o algoritmo guloso determinístico obteve resultados melhores do que o randomizado na instância maior, com cor máxima de 210. Isso se deve à ordenação aprimorada por grau combinado (direto + distância 2), que prioriza eficientemente os vértices mais restritos.

**Algoritmo Randomizado:** O algoritmo randomizado com $\alpha = 0.3$ apresentou desempenho inferior ao guloso (melhores soluções entre 221-225). Isso ocorre porque a aleatoriedade introduzida pelo valor de $\alpha$ pode prejudicar a escolha gulosa em grafos muito densos, onde as restrições são numerosas.

**Algoritmo Reativo:** O algoritmo reativo obteve os melhores resultados gerais, com cor máxima entre 202-205. Os valores de $\alpha$ mais conservadores ($\{0.02, 0.05, 0.10, 0.15\}$) permitiram que o algoritmo mantivesse escolhas mais gulosas enquanto ainda explorava o espaço de soluções. O mecanismo adaptativo convergiu consistentemente para $\alpha = 0.02$, indicando que escolhas quase-gulosas são mais eficazes para esta classe de instância.

# 6 Conclusão

Os experimentos demonstraram a eficácia das heurísticas para o problema L(p,q)-coloring, com resultados que variam conforme as características da instância e os parâmetros utilizados.

**Principais conclusões:**

1. O **algoritmo reativo** apresentou os melhores resultados em termos de qualidade de solução, alcançando cor máxima de 202 na instância r250.5.col, uma melhoria de 3.8% em relação ao guloso.

2. A **ordenação por grau combinado** (direto + distância 2) mostrou-se crucial para a qualidade das soluções gulosas.

3. Valores de $\alpha$ **mais conservadores** ($0.02-0.15$) são mais eficazes para grafos densos, contrariando a intuição de que maior aleatoriedade sempre beneficia a exploração do espaço de soluções.

4. O **trade-off tempo/qualidade** é evidente: o algoritmo guloso é aproximadamente 6700x mais rápido que o reativo, mas produz soluções 3.96% piores.

5. A **busca local** no algoritmo reativo contribui significativamente para a qualidade final das soluções.

**Trabalhos Futuros:**

- Testar instâncias maiores e de diferentes classes (grafos esparsos, bipartidos, etc.)

- Implementar outras meta-heurísticas como Simulated Annealing ou Algoritmos Genéticos
- Paralelizar as iterações do GRASP para reduzir o tempo de execução

# Referências


[1] Griggs, J. R., & Yeh, R. K. (1992). *Labelling graphs with a condition at distance 2*. SIAM Journal on Discrete Mathematics, 5(4), 586-595.

[2] Resende, M. G. C., & Ribeiro, C. C. (2003). *Greedy Randomized Adaptive Search Procedures*. Handbook of Metaheuristics, Kluwer Academic Publishers, 219-249.

[3] Prais, M., & Ribeiro, C. C. (2000). *Reactive GRASP: An Application to a Matrix Decomposition Problem in TDMA Traffic Assignment*. INFORMS Journal on Computing, 12(3), 164-176.