Universidade Estadual de Campinas
Instituto de Computação

Ana Paula dos Santos Dantas

Recoloração Convexa de Grafos

CAMPINAS
2019

Ana Paula dos Santos Dantas

Recoloração Convexa de Grafos

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestra em Ciência da Computação.

Orientador: Prof. Dr. Zanoni Dias
Coorientador: Prof. Dr. Cid Carvalho de Souza

Este exemplar corresponde à versão final da Dissertação defendida por Ana Paula dos Santos Dantas e orientada pelo Prof. Dr. Zanoni Dias.

CAMPINAS
2019

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

D235r

Dantas, Ana Paula dos Santos, 1996-
Recoloração convexa de grafos / Ana Paula dos Santos Dantas. – Campinas, SP : [s.n.], 2019.

Orientador: Zanoni Dias.
Coorientador: Cid Carvalho de Souza.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

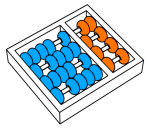
1. Recoloração convexa. 2. Otimização combinatória. 3. GRASP (Meta-heurística). 4. Programação linear inteira. I. Dias, Zanoni, 1975-. II. Souza, Cid Carvalho de, 1963-. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Convex graph recoloring
Palavras-chave em inglês:
Convex recoloring
Combinatorial optimization
GRASP (Metaheuristic)
Integer linear programming
Área de concentração: Ciência da Computação
Titulação: Mestra em Ciência da Computação
Banca examinadora:
Zanoni Dias [Orientador]
Rafael Crivellari Saliba Schouery
Simone de Lima Martins
Data de defesa: 27-09-2019
Programa de Pós-Graduação: Ciência da Computação

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0002-8831-0710
- Currículo Lattes do autor: http://lattes.cnpq.br/8815503877541145

Universidade Estadual de Campinas
Instituto de Computação



Ana Paula dos Santos Dantas

Recoloração Convexa de Grafos

Banca Examinadora:

- Prof. Dr. Zanoni Dias
  Instituto de Computação - Unicamp
- Profa. Dra. Simone de Lima Martins
  Instituto de Computação - UFF
- Prof. Dr. Rafael Crivellari Saliba Schouery
  Instituto de Computação - Unicamp

A ata da defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 27 de setembro de 2019

Dedicatória

Aos meus pais.

So let us wage, so let us wage a glorious struggle against illiteracy, poverty and terrorism, let us pick up our books and our pens, they are the most powerful weapons.

(Malala Yousafzai)

Agradecimentos

Agradeço à minha mãe, Ivanilda, e ao meu pai, Francisco, pelos sacrifícios que fizeram para que eu pudesse usufruir de uma boa educação, pelo apoio e carinho que sempre me foi dado durante toda a minha caminhada e por serem meu alicerce. Agradeço aos meus irmãos, Letícia e Filipe, que sempre estão ao meu lado quando preciso.

Agradeço aos Profs. Zanoni Dias e Cid C. de Souza, por terem confiado e acreditado em mim, pela excelente orientação durante esses dois anos. Agradeço por estarem sempre disponíveis para me ajudar, pela paciência e pelos conselhos. Aprendi muito com vocês. Não poderia ter escolhido melhores orientadores e sem vocês não teria chegado até aqui.

Agradeço ao Sandro, pelo companheirismo e carinho, por sempre estar ao meu lado e me oferecer um ombro para descansar.

Agradeço aos meus companheiros de laboratório e de disciplinas, pelos momentos que passamos juntos, os conselhos dados e dicas trocadas. Esses momentos me deram forças para chegar aqui.

Agradeço ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pela bolsa de pesquisa concedida por meio do processo processo 168051/2017-6, entre outubro de 2017 e julho de 2018.

Agradeço também à Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) e à Coordenadoria de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pela bolsa de pesquisa concedida, no âmbito do convênio FAPESP/CAPES, por meio do processo nº 2018/04760-3, Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), entre agosto de 2018 e setembro de 2019.

A todos que direta ou indiretamente me ajudaram a chegar aqui, o meu muito obrigada.

Resumo

Consideramos uma coloração de um grafo como uma função que define uma cor para todo vértice, independente de sua vizinhança. Desse modo, uma coloração é dita convexa se o conjunto formado por todos os vértices de mesma cor induz um subgrafo conexo. Dada uma coloração qualquer, o Problema de Recoloração Convexa busca pelo menor número de vértices que precisam mudar de cor, isto é, ser recoloridos, de modo que a coloração se torne convexa. Esse problema é mais comumente abordado na sua versão em árvores devido à sua origem no estudo de árvores filogenéticas. Neste trabalho tratamos de uma variante do problema em árvores cuja coloração é aplicada apenas nas folhas. Para essa versão do problema, fizemos experimentos com o modelo de Programação Linear Inteira (PLI) proposto para o problema em árvores por Chopra *et al.* [9]. Verificamos que o modelo também tem bons resultados para a versão do problema em árvores com apenas as folhas coloridas. Para a versão do problema em grafos gerais, propomos uma heurística baseada no GRASP (*Greedy Randomized Adaptive Search Procedure*). Executamos extensivos experimentos com essa heurística e usamos o modelo de Programação Linear Inteira proposto por Campêlo *et al.* [7] para verificar a qualidade das soluções do GRASP. A heurística GRASP recoloriu um número de vértices menor que o modelo em uma quantidade significativa de instâncias, quando ambos dispõem da mesma quantidade de recursos computacionais. Além disso, adaptamos a heurística e o modelo PLI para uma versão do problema de recoloração conhecida como Problema de Recoloração Convexa Restrita. Nessa versão, restringimos quais vértices podem ser recoloridos e quais podem apenas ter suas cores removidas. Criamos *benchmarks* de instâncias para todas as versões estudadas do problema e, sempre que necessário, usamos testes estatísticos para fundamentar as escolhas das melhores versões da heurística.

Abstract

We consider a coloring of a graph to be a function that assigns a color to a vertex, regardless of its neighborhood. In this sense, a coloring is said to be convex if every set formed by all of the vertices with the same color induces a connected subgraph. The Convex Recoloring Problem asks to find the minimum number of vertex recolorings needed to turn a coloring convex. This problem is most commonly treated in its version that considers trees due to its origins in the study of phylogenetic trees. We worked on a version of the problem in which the initial coloring is applied only on the leaves of the tree. For this version, we experimented with the mathematical model proposed for the problem on trees by Chopra *et al.* [9]. With these experiments, we found that the model also has good results for the version with the coloring only on the leaves. For the version of the problem on general graphs, we proposed a heuristic based on the Greedy Randomized Adaptive Search Procedure (GRASP) and used a mathematical model proposed by Campêlo *et al.* [7] to verify the quality of the solutions given by the GRASP heuristic. In these experiments, our heuristic recolored less vertices than the model in a significant number of instances, when given the same computational resources. We also propose adaptations for the GRASP to solve a version of the recoloring problem known as Minimum Restricted Recoloring Problem. In this version, we restrict which vertices can be recolored and which vertices can only have their color removed. We created sets of benchmark instances for all the versions of the problem, and performed statistical analysis to support our conclusions, when necessary.

Lista de Figuras

Sumário

Capítulo 1

Introdução

O problema de recoloração convexa tem suas origens na Biologia Computacional, mais especificamente no estudo de árvores filogenéticas. Uma árvore filogenética é uma estrutura que descreve o histórico da evolução de um conjunto de espécies. Nessa árvore, vértices folha representam espécies atuais e vértices internos representam seus antepassados hipotéticos.

Um conjunto de características comuns às espécies estudadas é usado para construir a árvore filogenética. Desse modo, é esperado que espécies com características semelhantes estejam próximas na árvore [21]. Cada característica usada na construção pode se manifestar em diferentes estados, contudo, cada espécie manifesta apenas um dos estados possíveis de cada característica.

Na versão mais simples do problema, uma árvore é construída considerando apenas uma única característica. Nesse caso, podemos agrupar as espécies em classes que apresentam o mesmo estado da característica estudada. Ao atribuir uma cor para cada classe, cada vértice da árvore terá uma única cor [19]. Chamamos essa atribuição de cores de *coloração*[1]. Após a construção de uma árvore filogenética, queremos saber o seu nível de adequação. Para isso existem duas medidas clássicas: a parcimônia e a compatibilidade.

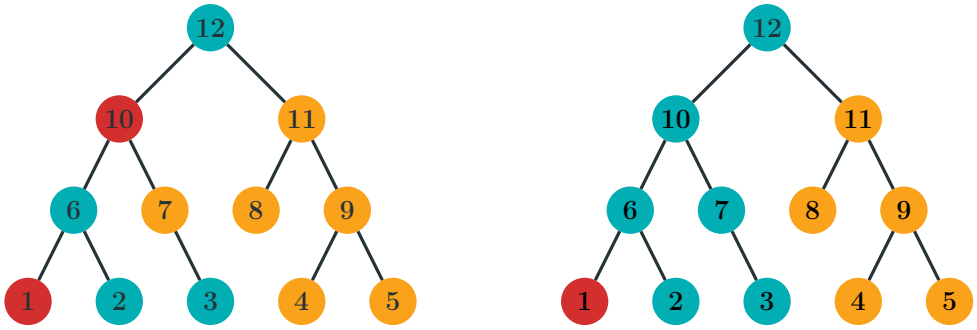
Pelo critério da parcimônia, uma árvore filogenética deve ter a menor soma possível dos pesos das arestas. O peso de uma aresta é definido como a soma do número de características diferentes nos vértices extremos da aresta [21]. Por exemplo, na versão simples, se a aresta liga dois vértices que apresentam o mesmo estado, ela tem custo 1, caso contrário, ela tem custo 2.

Já pelo critério da compatibilidade, uma árvore filogenética deve ter o maior número de características convexas. Uma característica é convexa se as espécies que apresentam o mesmo estado formam uma única subárvore, que usa apenas as arestas cujos dois extremos possuem o estado em questão. Usando esse critério no esquema de cores em uma árvore simples, todos os vértices que possuem a mesma cor induzem uma única subárvore [21].

Uma árvore filogenética é *ideal* se todas as características estudadas são convexas. Além disso uma árvore ideal é ótima sob os critérios da parcimônia e da compatibilidade e é chamada de filogenia perfeita [21].

A necessidade de que uma árvore filogenética seja ideal vem da suposição de que as

[1]Neste trabalho, a menos que esteja explícito, o termo coloração não se refere a uma coloração própria, que diz que dois vértices adjacentes não podem ter a mesma cor.



(a) árvore colorida (b) recoloração convexa

Figura 1.1: Árvore filogenética colorida. A Figura 1.1(a) mostra uma coloração de uma árvore filogenética que não é ideal e a Figura 1.1(b) mostra uma possível recoloração convexa para a árvore colorida anterior.

características deveriam evoluir sem a ocorrência de reversões ou convergências. Uma reversão indica que uma espécie readquiriu um estado de uma característica que seu ancestral direto perdeu. Já uma convergência indica que espécies distintas adquiriram um estado de uma característica que o seu ancestral comum mais recente não possui [29]. Veja o exemplo da Figura 1.1. Suponha que as árvores da Figura 1.1 são árvores filogenéticas, nas quais cada vértice representa uma espécie e as cores dos vértices são os estados de uma característica estudada. A árvore da Figura 1.1(a) não é ideal, já que existem classes que não induzem uma única subárvore, como a classe formada pelos vértices de cor azul (●). Nessa árvore podemos ver as duas características não desejadas durante o processo de evolução: reversão e convergência.

Podemos ver uma reversão no vértice 1, que tem a cor vermelha (●). Essa característica não foi mantida pelo seu antepassado direto, o vértice 6. Já a convergência pode ser vista nos vértices 7 e 11, que são coloridos com a cor amarela (●), mas seu antepassado mais recente, o vértice 12, não possui esse estado da característica. A árvore da Figura 1.1(b) é ideal, logo, não encontramos casos de reversões ou convergências.

Os critérios que medem a adequação da árvore filogenética, citados acima, não oferecem uma boa estimativa sobre a distância de uma árvore filogenética até a sua filogenia perfeita. Para isso Moran e Snir [29] definiram a ***distância de recoloração***, que indica o menor número de vértices que devem ter suas cores alteradas para que a árvore seja ideal, definindo assim o Problema de Recoloração Convexa de Árvores (*Convex Tree Recoloring* – CTR).

Na literatura, o CTR é tratado como um problema de grafos e o esquema de cores é usado para representar os estados das características nos vértices das árvores. Apesar de ter sua motivação em árvores, o problema pode ser generalizado para tratar grafos gerais, de modo que cada classe de cor deva induzir um subgrafo conexo, e não uma subárvore. Variações do problema são comuns tanto na classe do grafo, quanto na forma como a coloração é aplicada.

Por exemplo, o problema da Recoloração Convexa de Caminhos (*Convex Path Recoloring* – CPR) altera a classe do grafo. Quando a classe do grafo não é especificada, o problema é conhecido apenas como Recoloração Convexa (*Convex Recoloring* – CR). Já o

problema de Recoloração Convexa de Árvores com apenas Folhas Coloridas (*Convex Leaf Recoloring* – CLR) é uma variação do problema de recoloração convexa em árvores que restringe a coloração inicial, aplicando-a apenas nas folhas da árvore.

Neste trabalho, abordamos o problema de recoloração convexa em duas classes de grafos: árvores e grafos gerais. Para ambas as classes de grafos, focamos o nosso trabalho em investigar métodos exatos para o problema, propor instâncias e realizar experimentos computacionais. No contexto de árvores focamos na versão com apenas folhas coloridas. Investigamos um modelo de Programação Linear Inteira para o problema de Recoloração Convexa em Árvores e o aplicamos na versão com apenas folhas coloridas.

Já no contexto de grafos gerais, propomos uma heurística GRASP para o problema e também investigamos um modelo de Programação Linear Inteira. Além do problema de recoloração convexa, abordamos o problema de Recoloração Convexa Restrita, que possui aplicações em redes de computadores. Modificamos o modelo de Programação Linear Inteira e a heurística do CR para tratar essa versão.

A organização do restante deste trabalho é descrita a seguir. Apresentamos no Capítulo 2 os conceitos básicos que são usados nos capítulos seguintes. No Capítulo 3, descrevemos o trabalho desenvolvido com o Problema de Recoloração Convexa em Árvores com apenas Folhas Coloridas. No Capítulo 4 apresentamos o trabalho desenvolvido com o Problema de Recoloração Convexa em grafos gerais e no Capítulo 5 tratamos do problema de Recoloração Convexa Restrita. Por último, no Capítulo 6, fazemos nossas considerações finais e discutimos sobre trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Neste capítulo revisamos alguns conceitos usados neste trabalho. Apresentamos conceitos básicos sobre Teoria dos Grafos e sobre colorações. Também definimos as notações que são usadas no restante do texto. Em seguida, apresentamos a definição básica da meta-heurística GRASP, que é usada nos Capítulos 4 e 5.

Definimos um ***grafo não direcionado*** G como um par ordenado (V, E), sendo V um conjunto de vértices e E um conjunto de pares não ordenados de vértices, chamados de arestas, tal que $E \subseteq V \times V$. Dizemos que dois vértices u e v são vizinhos, ou adjacentes, se existe uma aresta $e = \{u, v\} \in E$. Também chamamos u e v de extremos da aresta e.

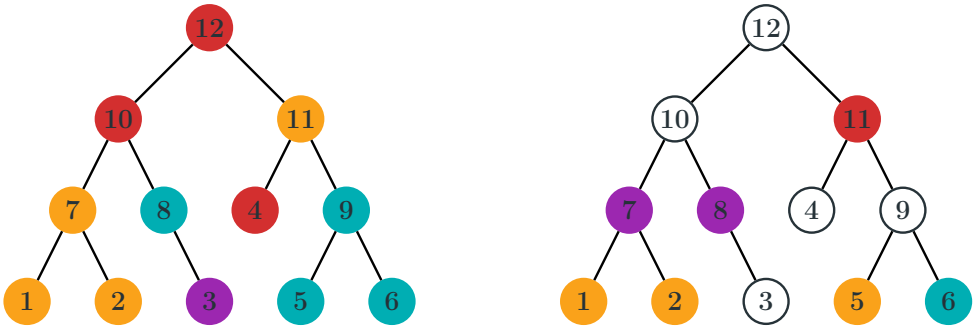
Um ***caminho*** entre dois vértices v_0 e v_k é uma sequência $\{v_0, e_1, v_1, \ldots, v_{k-1}, e_k, v_k\}$ de vértices v_i e arestas e_i, tal que $e_i = v_{i-1}v_i$ e não existem vértices repetidos, ou seja, para todo $i \neq j$ temos que $v_i \neq v_j$. Dizemos que um grafo G é ***conexo*** se, e somente se, existe um caminho entre quaisquer dois vértices de G. Um ***ciclo*** é uma sequência $\{v_0, e_1, v_1, \ldots, v_{k-1}, e_k, v_0\}$ de vértices v_i e arestas e_i, tal que $e_i = v_{i-1}v_i$ e o último vértice é igual ao primeiro. Uma ***árvore*** é um grafo conexo sem ciclos [4].

Seja S um subconjunto de vértices do grafo G, chamamos de subgrafo ***induzido*** por S, ou $G[S]$, o grafo cujo conjunto de vértices é o próprio conjunto S e o conjunto de arestas é definido como $\{e = uv | \ \forall e \in E$, tal que $u, v \in S\}$, isto é, formado pelas arestas do grafo original que possuem os dois extremos em S.

Usamos o termo ***coloração*** para designar a associação de uma cor a um vértice, independente de sua adjacência e chamamos de grafo colorido o par (G, C) formado pelo grafo G e pela coloração C dos vértices de G. Definimos uma coloração como uma função que recebe de entrada um vértice e retorna uma cor.

Representamos uma cor como um número inteiro, de modo que um conjunto de k cores é definido da seguinte forma: $\mathcal{C} = \{1, 2, \ldots, k\}$. Também usamos $\emptyset$ para representar a ausência de cor em um vértice. Uma ***classe de cor*** é o conjunto de todos os vértices que possuem a mesma cor. Chamamos uma cor de ***convexa*** se ela induz um subgrafo conexo, caso contrário, dizemos que a cor é ***ruim***.

Se uma coloração $C : V \to \mathcal{C}$ define uma cor para todos os vértices do grafo, então chamamos essa coloração de ***coloração total***. Chamamos de ***coloração parcial*** uma coloração que admite vértices que não possuem cor, ou seja, associados a $\emptyset$, de modo que a coloração é definida por $C : V \to \mathcal{C} \cup \{\emptyset\}$. Chamamos de ***coloração convexa*** uma coloração na qual cada classe de cor induz um subgrafo conexo. Uma ***coloração boa*** é



(a) coloração total não convexa

(b) coloração parcial não convexa

(c) coloração boa

(d) coloração total convexa

Figura 2.1: Exemplos dos tipo de colorações apresentados com variações na totalidade e convexidade.

uma coloração que é parcial e convexa [5].

Uma coloração total pode ser obtida a partir de uma coloração boa em tempo linear mantendo a convexidade. Para tal, basta encontrar um caminho de vértices descoloridos entre um vértice x sem cor, $C(x) = \emptyset$, e um vértice colorido y tal que $C(y) = c,\ c \in \mathcal{C}$; definir a cor dos vértices visitados no caminho como c, e repetir os passos anteriores enquanto houver vértice sem cor [5].

Apresentamos exemplos para os tipos de coloração na Figura 2.1. Os vértices brancos representam vértices sem cor, isto é, $C(v) = \emptyset$. Nas Figuras 2.1(a) e 2.1(b) temos exemplos de colorações não convexas, a primeira é uma coloração total e a segunda é uma coloração parcial. Na Figura 2.1(c), temos uma coloração parcial na qual todas as classes de cor induzem subgrafos conexos, logo a coloração é boa. Já na Figura 2.1(d), também apresentamos uma coloração convexa, entretanto, essa é uma coloração total obtida a partir da coloração anterior, usando o algoritmo apresentado por Campêlo *et al.* [5].

Uma ***recoloração*** C' de um grafo colorido é uma coloração no mesmo conjunto de vértices. Quando criamos uma recoloração, queremos alcançar uma certa propriedade no grafo que no nosso caso é a convexidade. Podemos medir a ***distância de recoloração*** usando uma função $w : V \to \mathbb{Q}_{\geq 0}$, que indica quanto custa mudar a cor de um determinado vértice. Na versão não ponderada do problema, o custo de mudança de cor é unitário.

Dados um grafo colorido (G, C) e uma recoloração C', definimos o conjunto $R_C(C')$ como $\{v \in V \mid C(v) \neq \emptyset \text{ e } C(v) \neq C'(v)\}$, isto é, o conjunto dos vértices que tiveram sua cor alterada na recoloração. O conjunto $R_C(C')$ e a função de custo w são usados para

calcular o custo total de uma recoloração.

Note que um vértice v não pertence ao conjunto $R_C(C')$ se ele não tinha uma cor definida por C ($C(v) = \emptyset$), mas tem uma cor definida por C' ($C'(v) = c, c \in \mathcal{C}$). Como o conjunto R_C é usado para calcular o custo de uma recoloração, isso implica que colorir um vértice que antes estava sem cor tem custo zero.

A maior parte deste trabalho é dedicada à descrição de um algoritmo baseado na meta-heurística *Greedy Randomized Local Search Procedure* (GRASP) para o problema de recoloração. A meta-heurística GRASP foi proposta por Feo e Resende [14], que usaram o método para resolver uma versão do problema de cobertura por conjuntos (*set covering*).

Desde então, a meta-heurística vem sendo usada em diferentes problemas, como alocação e roteamento [18]. O GRASP também já foi aplicado em problemas de Teoria dos Grafos, como o problema de encontrar o conjunto independente máximo de um grafo [16] e o problema de coloração própria em grafos esparsos [24].

O GRASP é um procedimento iterativo que consiste em duas fases em cada iteração: construção da solução e busca por um ótimo local. As duas fases são executadas em sequência e são repetidas até que um critério de parada seja atendido. O número total de iterações é um critério de parada comumente usado. No Algoritmo 1 representamos um esquema do GRASP. Esse esquema é baseado na versão apresentada por Feo e Resende [15] e adaptado para problemas de recoloração convexa. Essa versão do GRASP é a mais simples, com iterações independentes, sem mutações ou outras meta-heurísticas combinadas.

Algoritmo 1 GRASP – *Greedy Randomized Adaptive Search Procedure*

```
1: função GRASP((G, C), k, α)
2:     melhorSolução ← ∅
3:     para i = 1, 2, ..., k faça
4:         solução ←SOLUÇÃOGULOSA((G, C), α)
5:         solução ←BUSCALOCAL(G, C, solução)
6:         melhorSolução ← MIN(melhorSolução, solução)
7:     fim para
8:     devolve melhorSolução
9: fim função
```

Como entrada para o Algoritmo 1, são passados um grafo colorido (G, C), um número de iterações k e um valor de α, tal que $0 < \alpha \leq 1$, que é usado durante a construção da solução. Na Linha 2 a melhor solução é inicializada com $\emptyset$, que tem $+\infty$ como custo de recoloração. O laço definido nas Linhas 3 à 7 é repetido k vezes. Dentro desse laço, uma solução gulosa aleatorizada é construída (Linha 4); uma busca local na vizinhança dessa solução é feita (Linha 5); e, caso a solução atual seja a melhor criada até o momento, a variável *melhorSolução* é atualizada.

Usamos a função MIN na Linha 6, pois, estamos considerando o problema de recoloração convexa como um problema de minimização do custo de recoloração. Desse modo, a variável *melhorSolução* tem a solução de menor custo criada nas k iterações do GRASP no final do processo, e esta é a resposta devolvida pelo algoritmo.

O algoritmo que constrói uma solução gulosa na Linha 4 também é um processo

iterativo e é responsável pelos três elementos-chave do GRASP, já que a solução construída por ele deve ser gulosa (*greedy*), aleatorizada (*randomized*) e adaptável (*adaptive*). Esses componentes são detalhados a seguir.

Para construirmos uma solução gulosa, a cada iteração temos uma solução parcial e os elementos que podem ser adicionados a essa solução são ordenados de acordo com um critério guloso. Esse critério mede o quão bom é um elemento, comparado com as outras opções disponíveis naquele ponto da construção da solução.

Com os elementos candidatos ordenados, o próximo passo é criar uma lista restrita de candidatos (RCL, do inglês *Restricted Candidate List*). Essa lista contém apenas uma porcentagem α dos melhores candidatos da lista anterior. Em seguida um elemento da RCL é escolhido aleatoriamente para entrar na solução.

Por último, sempre que um novo elemento é inserido o estado da solução parcial muda. Assim, cada iteração do algoritmo que cria uma solução gulosa aleatorizada afeta as iterações seguintes. Note que as iterações do GRASP, nas Linhas 3 a 7 são independentes na versão simples do algoritmo. Isto é, a criação da i-ésima solução não afeta a criação de uma j-ésima solução, com $0 \leq i, j \leq k$ e $i \neq j$.

Capítulo 3

Recoloração Convexa de Árvores

Este capítulo aborda o problema de Recoloração Convexa de Árvores e sua variação Recoloração Convexa de Árvores com apenas Folhas Coloridas. Na Seção 3.1, introduzimos os problemas, além de fazermos um levantamento da literatura. Na Seção 3.2, fornecemos em detalhes a abordagem da literatura que foi utilizada nos experimentos deste capítulo. Na Seção 3.3, descrevemos os experimentos e discutimos seus resultados. Por último, na Seção 3.4, apresentamos as conclusões e listamos sugestões de trabalhos futuros.

3.1 Introdução

O problema de Recoloração Convexa de Árvores é a versão mais estudada dos problemas de recoloração convexa e é conhecido na literatura como *Convex Tree Recoloring* (CTR). O CTR recebe como entrada uma árvore colorida e retorna uma coloração boa com custo mínimo. A Definição 3.1 apresenta o CTR formalmente, onde são especificados a entrada, o objetivo e a saída do problema.

Definição 3.1 RECOLORAÇÃO CONVEXA DE ÁRVORES – CTR

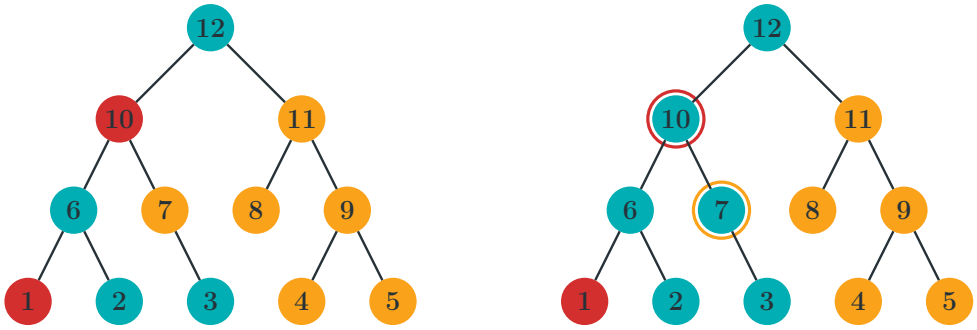
Entrada: Uma árvore $T = (V, E)$, um conjunto de cores $\mathcal{C}$, uma coloração parcial $C : V \to \mathcal{C} \cup \{\emptyset\}$ e uma função de custo w.

Objetivo: Encontrar uma coloração boa C' com custo mínimo, $\min(\sum_{v \in R_C(C')} w(v))$.

Saída: Uma coloração boa C'.

Usamos a Figura 3.1 para exemplificar a definição do CTR. Suponha que a árvore colorida da Figura 3.1(a) é dada como entrada para o problema, juntamente com um conjunto de cores $\mathcal{C} = \{$●, ●, ●$\}$ e uma função de custo uniforme que atribui custo 1 para todos os vértices. Na Figura 3.1(b), temos uma recoloração convexa para a entrada fornecida, onde os vértices que foram recoloridos são marcados com um círculo externo na sua cor original. Essa recoloração troca a cor de dois vértices (7 e 10) e tem custo 2.

Moran e Snir [29] provaram que o CTR é NP-difícil usando uma redução do 3-SAT para o problema. As abordagens para resolver o CTR presentes na literatura são variadas. Moran e Snir [28] e Kanj e Kratsch [23] propuseram algoritmos exatos com tempo



(a) coloração não convexa (b) recoloração convexa mínima

Figura 3.1: Recoloração Convexa em Árvores. A Figura 3.1(a) apresenta uma instância do problema CTR. A Figura 3.1(b) apresenta recoloração convexa mínima para a instância da figura anterior.

exponencial para o problema. Posteriormente, Ponta *et al.* [32] desenvolveram melhorias na complexidade desses algoritmos.

Moran e Snir [28] apresentaram uma 3-aproximação para o problema. Esse fator de aproximação foi melhorado para $(2 + \epsilon)$ por Bar-Yehuda, Feldman e Rawitz [2]. O algoritmo de aproximação proposto por Bar-Yehuda, Feldman e Rawitz [2] é dividido em duas partes, a primeira com complexidade de tempo de $\mathcal{O}(n^2)$ e a segunda com complexidade $\mathcal{O}(n^2 + nk^2 2^k)$, onde n é o número de vértices e k é um parâmetro de acurácia definido por $k = \lceil \frac{2}{\epsilon} \rceil + 1$.

A primeira modelagem linear inteira para o CTR foi proposta por Campêlo *et al.* [5]. Esse modelo tem um número polinomial de variáveis e um número exponencial de restrições. A formulação é genérica e pode ser usada com grafos gerais. Além disso, Campêlo *et al.* [5] reportaram experimentos computacionais com o modelo proposto usando árvores filogenéticas disponíveis na base TreeBASE[1].

Chopra *et al.* [9] desenvolveram um modelo de programação linear inteira com número linear de restrições e de variáveis. Foram fornecidas provas de que essa formulação é estritamente mais forte que a formulação de Campêlo *et al.* [5]. Além disso, os autores reportaram em experimentos computacionais que o modelo conseguiu resolver instâncias com até 5000 vértices, quando o número de cores é pequeno.

Em uma continuação do trabalho anterior, Chopra *et al.* [8] trataram o problema com um modelo de geração de colunas. Esse novo modelo foi capaz de resolver todas as instâncias com mais de 1000 vértices que o modelo anterior não conseguia, mas para instâncias pequenas não obteve resultados antes do limite de tempo.

Um caso específico do CTR que também é bastante estudado na literatura é a Recoloração Convexa de Caminhos (ou *Convex Path Recoloring* – CPR). Nessa versão, consideramos um caminho como uma árvore com exatamente duas folhas. O CPR tem uma 2-aproximação [28], além de um modelo de programação linear inteira apresentado por Lima e Wakabayashi [26]. Lima [25] desenvolveu um estudo sobre as facetas do poliedro e combinações de algoritmos *branch-and-cut* com programação dinâmica que obtiveram bons resultados.

[1]Disponível em http://treebase.org/treebase-web

Quando existe um limite k no tamanho das classes de cores, o problema é conhecido como k-CTR. Essa notação também é usada para outras versões dos problemas de recoloração convexa. Por exemplo, adicionando ao CPR a restrição de que cada classe de cor possui no máximo dois vértices, temos a versão conhecida como 2-CPR. Para essa versão, existe um algoritmo com fator de aproximação 5/4 [3].

Outra variação do CTR proíbe que a coloração inicial defina uma cor para os vértices internos da árvore, que é conhecida na literatura como Recoloração Convexa de Árvores com apenas Folhas Coloridas (ou *Convex Leaf-colored tree Recoloring* – CLR). Essa é uma modificação natural para o problema CTR, que surge a partir da forma como as árvores filogenéticas são construídas. Já que os pesquisadores conhecem apenas as espécies que estão representadas nas folhas dessas árvores, podemos desconsiderar as suposições feitas sobre os antepassados removendo a coloração desses vértices. A Definição 3.2 enuncia essa variação do problema.

Definição 3.2 RECOLORAÇÃO CONVEXA DE ÁRVORES COM APENAS FOLHAS COLORIDAS

Entrada: Uma árvore $T = (V, E)$, um conjunto de cores $\mathcal{C}$, uma coloração parcial $C : V \to \mathcal{C} \cup \{\emptyset\}$, tal que $C(v) \neq \emptyset$, se e somente se v é uma folha e uma função de custo w.

Objetivo: Encontrar uma coloração boa C' com custo mínimo, $\min(\sum_{v \in R_C(C')} w(v))$.

Saída: Uma coloração boa C'.

Note que o objetivo e a saída do CLR são iguais aos do CTR (Definição 3.1), apenas a coloração da entrada é diferente. É nesse ponto onde é imposta a restrição na coloração inicial que diferencia os dois problemas. Ilustramos a definição do CLR na Figura 3.2.

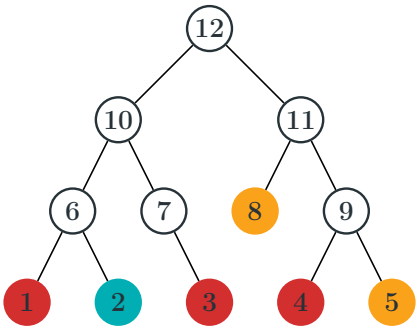
Seja o grafo colorido da Figura 3.2(a) uma instância para o problema CLR, onde vértices com a cor branca representam vértices que não possuem uma cor atribuída a ele. Nas duas figuras seguintes, Figura 3.2(b) e Figura 3.2(c), temos possíveis recolorações convexas dessa entrada. Supondo novamente que o custo de recoloração seja uniforme, ambas as recolorações possuem o mesmo custo.

O problema CLR também é NP-difícil [29]. Contudo, Kanj e Kratsch [23] provaram que se o tamanho de cada classe de cor é menor ou igual a 3, o problema pode ser resolvido em tempo polinomial com uma redução do problema de conjuntos independentes em grafos cordais. Essa versão do problema é conhecida como 3-CLR.

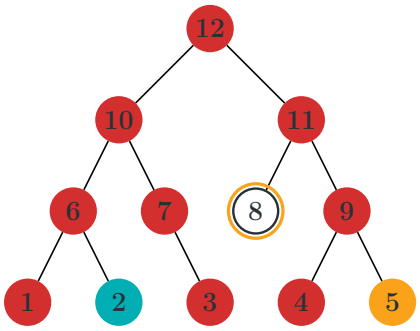
Bachoore e Bodlaender [1] apresentaram um algoritmo linear que testa se uma coloração parcial das folhas pode ser estendida para uma coloração total com custo 0, ou seja, sem a necessidade de trocar as cores dos vértices folhas. A Figura 3.3 exemplifica o caso identificado pelo algoritmo.

Assumindo a coloração inicial da Figura 3.3(a), podemos obter uma recoloração convexa com custo 0, ou seja, que não recolore vértices folhas. Essa extensão é apresentada na Figura 3.3(b). Vale lembrar que, recolorir um vértice que não tem cor na coloração inicial tem custo 0.

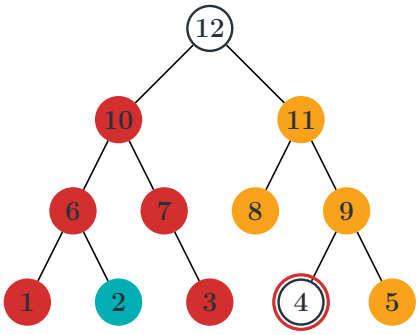
A literatura atual não apresenta métodos de resolução específicos para o CLR. Como o CLR é um caso particular do CTR, investigamos neste capítulo se os métodos de resolução



(a) coloração não convexa

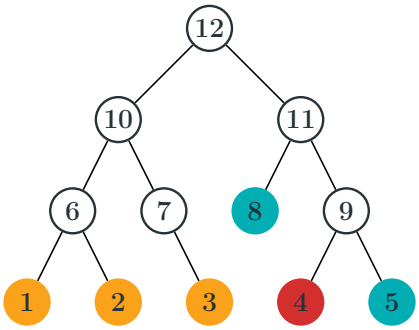


(b) recoloração convexa mínima

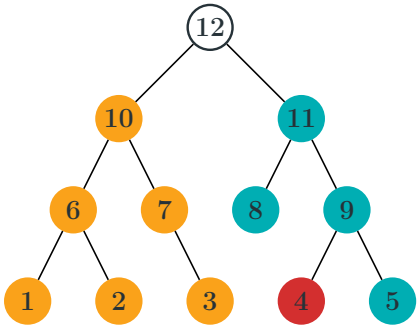


(c) recoloração convexa mínima

Figura 3.2: Recoloração Convexa de Árvores com apenas Folhas Coloridas. A Figura 3.2(a) apresenta uma instância do problema CLR. As Figuras 3.2(b) e 3.2(c) apresentam possíveis recolorações convexas, ambas possuem custo mínimo.



(a) coloração não convexa



(b) recoloração convexa mínima

Figura 3.3: Exemplo de extensão da coloração com custo 0. A Figura 3.3(a) apresenta uma instância do problema CLR. A Figura 3.3(b) mostra uma extensão da recoloração inicial que é convexa.

para o CTR são adequados, isto é, possuem um bom desempenho para a resolução de instâncias do CLR. As seções seguintes abordam essa questão.

3.2 Modelo PLI para o CTR

Na seção anterior fizemos um levantamento dos métodos de resolução do problema de recoloração convexa em árvores. Dentre as abordagens citadas escolhemos o modelo proposto por Chopra *et al.* [9], pois este apresentou os melhores resultados computacionais nos experimentos reportados na literatura. Esse modelo é descrito a seguir (Equações (3.1) – (3.6)).

O modelo possui duas variáveis de decisão x_{ut} e y_{et}, ambas binárias. A variável x_{ut} recebe o valor 1 quando o vértice u for colorido com a cor t e 0 caso contrário. Analogamente, a variável y_{et} recebe o valor 1 quando a aresta e for colorida com a cor t e 0 caso contrário. Esse modelo define uma cor para as arestas que ajudam a fazer com que a subárvore associada a uma dada cor seja conexa. Os conjuntos $\mathcal{C}, V, E$ são conjuntos de cores, vértices e arestas, respectivamente.

A constante k indica o número de cores na coloração inicial, ou seja, o tamanho do conjunto $\mathcal{C}$. As contantes $w(t, v)$ são definidas como $w(v)$ se a cor do vértice v é t na coloração inicial ($C(v) = t$), e 0 caso contrário.

$$\text{C-CTR} \quad \text{maximize} \sum_{t=1}^{k} \sum_{u \in V} w(u,t) x_{ut} \tag{3.1}$$

$$\text{sujeito a} \quad \sum_{t=1}^{k} x_{ut} \leq 1 \qquad \forall\, u \in V \tag{3.2}$$

$$\sum_{u \in V} x_{ut} - \sum_{e \in E} y_{et} \leq 1 \qquad \forall\, t \in \mathcal{C} \tag{3.3}$$

$$\left.\begin{aligned} -\, x_{ut} + y_{uvt} &\leq 0 \\ -\, x_{vt} + y_{uvt} &\leq 0 \end{aligned}\right\} \qquad \forall\, \{u,v\} \in E\,, \forall\, t \in \mathcal{C} \tag{3.4}$$

$$x_{ut} \in \{0,1\} \qquad \forall\, u \in V\,, \forall\, t \in \mathcal{C} \tag{3.5}$$

$$y_{et} \in \{0,1\} \qquad \forall\, e \in E\,, \forall\, t \in \mathcal{C} \tag{3.6}$$

Na Equação (3.1), temos a função objetivo do modelo, que maximiza o número de vértices que mantém a cor original. Essa maximização é equivalente a minimizar o número de vértices recoloridos. As Desigualdades (3.2) restringem que cada vértice pode receber no máximo uma única cor, admitindo que a recoloração final não seja total. As Desigualdades (3.3) limitam o número de arestas ao número de vértices com a cor t menos um. Essas equações remetem à propriedade de árvores que diz que o número de arestas é igual ao número de vértices menos um.

Entretanto, essas restrições não garantem que a subárvore formada pelos vértices de mesma cor seja conexa. Esse problema é resolvido pela combinação da função objetivo e das Desigualdades (3.4), que restringem que uma aresta seja colorida com a cor t apenas se

os dois vértices das pontas estiverem coloridos com a cor t. Por último, as Equações (3.5) e (3.6) definem que as variáveis de decisão x e y sejam binárias.

3.3 Experimentos Computacionais

Nesta seção, reportamos os experimentos realizados com o modelo de programação linear inteira proposto por Chopra *et al.* [9], descrito anteriormente. O modelo foi implementado usando a linguagem matemática `OPL` da IBM e o resolvedor de programação linear inteira usado foi o `IBM CPLEX` versão 12.8.

Os experimentos foram executados em uma máquina com CPU Intel(R) Core(TM) i7-4790K com 8 *cores* de 4.00GHz, 16GB de memória RAM e sistema operacional ArchLinux, versão 4.2.5-1-ARCH. As funções de *presolve* e de paralelismo do `CPLEX` foram desativadas para fins de replicação dos experimentos reportados por Chopra *et al.* [9].

Comparando Instâncias do CLR e CTR

Nesta seção, detalhamos o primeiro experimento realizado com o modelo de Chopra *et al.* [9]. Criamos as instâncias usando o procedimento proposto por Campêlo *et al.* [5], que gera instâncias com árvores enraizadas. A metodologia consiste em criar uma coloração convexa inicial e depois trocar as cores de alguns vértices aleatórios. Essas duas fases são descritas a seguir.

Na primeira fase, começamos colorindo o vértice raiz da árvore com a cor 1. Em seguida, cada filho pode receber a cor do pai ou uma cor diferente e que ainda não foi usada na construção. Isso é decido aleatoriamente, com probabilidade $1 - p_c$ do filho manter a cor do pai. Usamos p_c, do inglês *probability of change*, para indicar a probabilidade de inserir uma nova cor na coloração. Aplicamos esse passo recursivamente até que todos os vértices da árvore tenham sido coloridos.

Note que, se um vértice filho muda de cor, aumentamos o número de cores usadas na coloração em uma unidade. Logo, valores mais altos para a probabilidade de troca implicam em valores mais altos no tamanho do conjunto de cores da instância. Ao final desse passo, temos a garantia de que a coloração não vai induzir uma subárvore desconexa, pois, um vértice ou está colorido com a cor do seu pai, ou está colorido com uma cor que ainda não foi usada.

Com uma coloração convexa criada, a segunda fase itera sobre os vértices da árvore de modo a inserir ruídos na coloração. Essa fase também faz uso de um parâmetro aleatório p_n, do inglês *probability of noise*, que indica a probabilidade de um vértice receber uma cor aleatória dentre as que já foram usadas na fase anterior. Assim, um vértice mantém a sua cor inicial com probabilidade $1 - p_n$.

Essa metodologia foi usada por Campêlo *et al.* [5] e por Chopra *et al.* [9] na criação das instâncias para o CTR que foram reportadas em seus respectivos trabalhos. Os autores nos forneceram essas instâncias e selecionamos as 100 maiores árvores dos dois conjuntos para usar nesses experimentos. O número de vértices das árvores escolhidas variam entre 524 e 2632. Vale notar que essas instâncias cedidas são árvores filogenéticas obtidas da base de dados TreeBASE.

Com cada uma das 100 árvores, utilizamos a metodologia proposta por Campêlo *et al.* [5] para criar 10 instâncias com cada par (p_c, p_n), tal que $p_c \in \{0.005, 0.05, 0.5\}$ e $p_n \in \{0.25, 0.50, 0.75\}$, totalizando 9000 instâncias para o problema CTR. A partir dessas instâncias, geramos instâncias para o CLR removendo a coloração inicial dos vértices internos.

Executamos o modelo CTR com os dois conjuntos de instâncias limitando o tempo de execução em 2 horas. Obtivemos resultados similares aos reportados por Chopra *et al.* [8] para o conjunto de instâncias CTR. Listamos na Tabela 3.1 a quantidade de instâncias do CTR que atingiram o limite de tempo de execução.

Todas as instâncias $p_c = 0.005$ e $p_c = 0.05$ foram resolvidas antes do tempo limite e, por isso, as instâncias listadas têm $p_c = 0.5$. A coluna **Instância** contém os identificadores das árvores e a coluna **n** indica seus respectivos números de vértices. As colunas seguintes, $\mathbf{p_n = 0.25}$, $\mathbf{p_n = 0.50}$ e $\mathbf{p_n = 0.75}$, são nomeadas com os valores de p_n e mostram o número de instâncias que não terminaram a execução antes de duas horas.

Instância	**n**	$\mathbf{p_n = 0.25}$	$\mathbf{p_n = 0.50}$	$\mathbf{p_n = 0.75}$
Tr69195	1838	6	6	7
Tr46272	2025	4	4	5
Tr57261	2387	7	7	6
Tr60915	2409	6	7	7
Tr73427	2632	7	9	9

Tabela 3.1: Instâncias CTR que atingiram o tempo limite com valor de $p_c = 0.5$.

As instâncias da tabela possuem os maiores números de cores, pois, $p_c = 0.5$ implica em muitas trocas de cores nos vértices filhos. Além disso, essas também são as 5 maiores árvores do conjunto. Para a maior árvore (Tr73427), que possui 2632 vértices, apenas 5 das 30 instâncias com $p_c = 0.5$ foram resolvidas. Ao analisarmos os resultados para as instâncias do CLR, constatamos algo diferente, pois, todas as instâncias terminaram a execução antes do tempo limite.

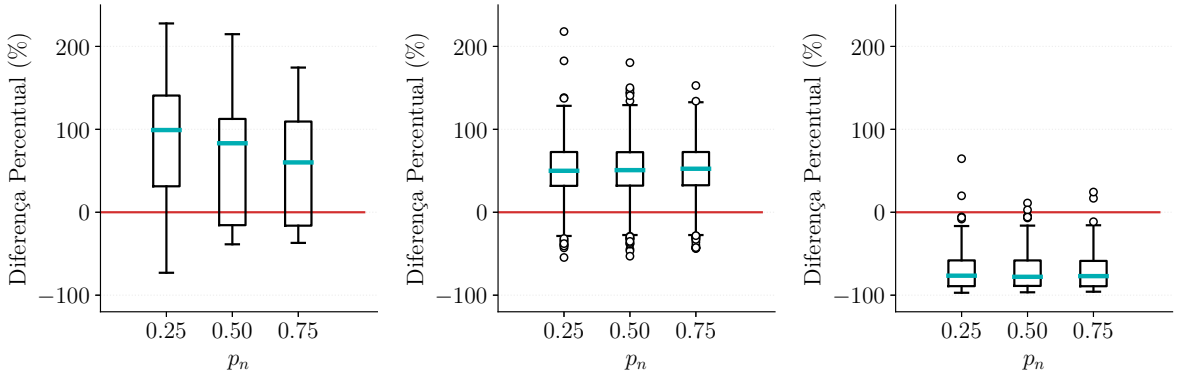
Selecionamos as instâncias cuja diferença no tempo de execução das duas versões foi de pelo menos um segundo para comparar os tempos de execução. Consideramos que uma diferença menor que um segundo pode ter sido causada por fatores externos à resolução da instância.

Na Tabela 3.2 temos o número de instâncias que tiveram uma diferença maior ou igual a um segundo em cada subconjunto (p_c, p_n). Ao filtrarmos as instâncias, perdemos as instâncias menores de alguns conjuntos, por isso informamos ao lado do número de instâncias o novo intervalo dos números de vértices de cada conjunto. Apenas duas instâncias com $p_c = 0.5$ tiveram diferença de tempo menor que um segundo. Fazendo uma análise mais detalhada, percebemos que muitas das instâncias removidas foram resolvidas em menos de 1 segundo.

Comparamos o tempo de execução das instâncias do CTR e do CLR usando a proporção do tempo das instâncias CLR (t_{CLR}) com relação ao tempo da instância correspondente

$\mathbf{p_c}$	$\mathbf{p_n = 0.25}$	$\mathbf{p_n = 0.50}$	$\mathbf{p_n = 0.75}$
0.005	39, $n \in [806, 2632]$	40, $n \in [1441, 2632]$	40, $n \in [855, 2632]$
0.050	363, $n \in [524, 2632]$	356, $n \in [583, 2632]$	353, $n \in [547, 2632]$
0.500	1000, $n \in [524, 2632]$	998, $n \in [524, 2632]$	1000, $n \in [524, 2632]$

Tabela 3.2: Instâncias com diferença de pelo menos 1 segundo no tempo de execução das duas versões do problema.



(a) $p_c = 0.005$ (b) $p_c = 0.05$ (c) $p_c = 0.5$

Figura 3.4: Distribuição das diferenças percentuais das instâncias do CTR e do CLR.

do CTR (t_{CTR}). Calculamos essa proporção com a seguinte fórmula:

$$\frac{t_{\text{CLR}} - t_{\text{CTR}}}{t_{\text{CTR}}}. \tag{3.7}$$

Criamos os gráficos da Figura 3.4 com os valores das proporções de tempo das instâncias da Tabela 3.2. Nesses gráficos, os valores de p_n estão representados no eixo x. No eixo y, temos os valores das proporções calculadas com a fórmula acima para todas as árvores. Desse modo, quando o valor de y é positivo, a diferença $t_{\text{CLR}} - t_{\text{CTR}}$ é positiva e a versão da instância para o problema CLR precisou de mais tempo para executar.

As Figuras 3.4(a), 3.4(b) e 3.4(c) indicam os valores para as instâncias com valor de $p_c = 0.005$, $p_c = 0.05$ e $p_c = 0.5$, respectivamente. Os dois primeiros gráficos mostram que a maioria das instâncias do CLR usou mais tempo de computação. Em alguns casos foi preciso quase o dobro do tempo, isto é, o valor de y está próximo de 200%.

Já o terceiro gráfico apresenta um comportamento diferente dos demais, isto é, as instâncias do CLR precisaram menos tempo para serem resolvidas em quase todos os casos. Note que essas são as instâncias que possuem o maior número de cores e em alguns casos a computação não terminou antes do tempo limite (Tabela 3.1). Por esses motivos, resolvemos analisar esse grupo de instâncias com mais detalhes.

Primeiro, verificamos que no conjunto CLR não existem instâncias cujas colorações iniciais poderiam ser estendidas para uma coloração convexa com custo 0. Em seguida, verificamos qual o impacto no número de cores presentes na coloração ao remover a coloração dos vértices internos. Com $p_c = 0.5$, percebemos que entre 20% e 40% das cores

que aparecem na versão CTR não são usadas na versão do CLR. Ou seja, essas cores eram usadas apenas nos vértices internos das árvores, o que pode explicar a diferença no tempo de execução.

Novas Instâncias para o CLR

No segundo experimento com o modelo CTR, verificamos seu desempenho com instâncias específicas para o CLR. Definimos um conjunto de 10 árvores binárias distintas com 999 vértices, sendo 500 destes folhas. Essas árvores são subárvores das listadas na Tabela 3.1. Para cada árvore criamos dois conjuntos de instâncias com diferentes abordagens para a colorações das folhas. Em todos os conjuntos, variamos o número k de cores de 2 a 9 com incrementos de uma unidade e de 10 a 500 com incrementos de 5 unidades, resultando em 107 valores para k.

No conjunto 1, para cada par (árvore, k), criamos uma lista com as folhas, onde as mesmas aparecem na ordem que são visitadas em uma DFS. Em seguida, definimos que a i-ésima folha tem a cor $(i \mod k) + 1$, com k sendo o número máximo de cores da instância. Por exemplo, com $k = 2$ a primeira folha recebe a cor 1, a segunda recebe a cor 2, a terceira recebe a cor 1, até que todas as folhas tenham recebido uma cor.

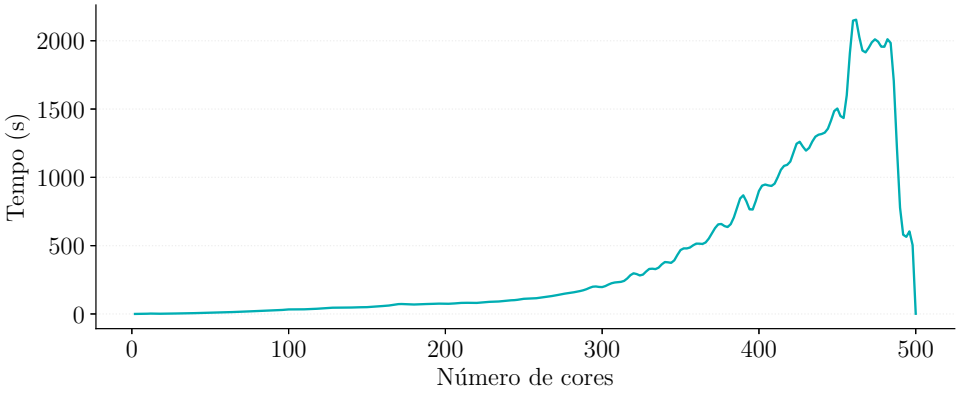
No conjunto 2, para cada par (árvore, k) criamos 10 instâncias. Visitamos cada folha, também na ordem em que aparecem na DFS, e escolhemos uma cor aleatória dentre as k disponíveis na instância. Nesse conjunto usamos uma distribuição uniforme para a escolha do número aleatório.

A Figura 3.5 mostra o tempo médio de execução para cada conjunto de instâncias. Em todos os gráficos, o eixo x indica o número de cores da instância e o eixo y indica a média do tempo de execução em segundos para todas as instâncias com o respectivo valor de k. Nesses experimentos também usamos o tempo limite de duas horas.

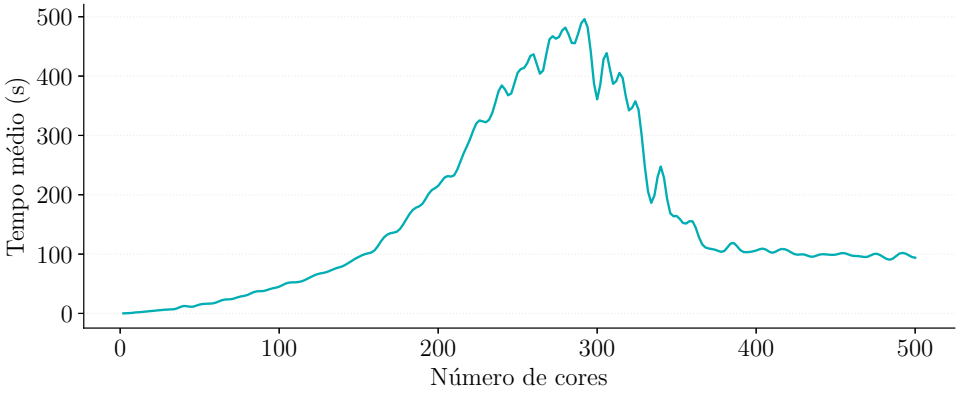
Os gráficos da Figuras 3.5(a) e 3.5(b) são referentes às instâncias dos conjuntos 1 e 2, respectivamente. Note que, os valores máximos em todos os gráficos são bem abaixo do limite de duas horas (ou 7200 segundos), o que significa que, mais uma vez, o modelo CTR teve um bom desempenho com instâncias do CLR.

Além disso, podemos observar algo interessante nesses gráficos, em especial no gráfico da Figura 3.5(a). As instâncias com $k \geq 170$ possuem no máximo 3 vértices por classe de cor, logo são instâncias que podem ser resolvidas em tempo polinomial [23]. O tempo de computação só começa a crescer de maneira mais acentuada bem depois desse valor.

O gráfico da Figuras 3.5(b) e 3.5(a) possuem comportamentos distintos, embora seja esperado que os tamanhos das classes de cores de cada instância seja igual. Verificamos a quantidade de cores usadas nas colorações do conjunto 2 e percebemos que, nas instâncias com $k > 250$, o número de classes de cores que de fato aparecem na coloração também diminuiu um pouco, em relação ao número esperado. Antes desse ponto, os tempos de execução são bem parecidos em ambos os conjuntos, note que as escalas dos eixos y dos gráficos são diferentes. Além disso, no conjunto 2 as classes de cores que possuem mais de um vértice não estão obrigatoriamente a uma distância fixa, isto é, elas podem estar bem próximas.



(a) conjunto 1



(b) conjunto 2

Figura 3.5: Experimento com instâncias do CLR. Nos gráficos nas Figuras 3.5(a) e 3.5(b) temos o tempo médio de execução das instâncias específicas para o problema CLR. Cada gráfico é referente a um tipo de coloração inicial apenas das folhas das árvores.

3.4 Conclusões

Neste capítulo, realizamos um levantamento da literatura existente para os problemas de Recoloração Convexa em Árvores (CTR) e Recoloração Convexa em Árvores com apenas Folhas Coloridas (CLR). Replicamos os experimentos do trabalho de Chopra *et al.* [9]. Desenvolvemos novos experimentos com o modelo CTR, que foi proposto por Chopra *et al.* [9]. Desenvolvemos instâncias para o CLR baseadas no procedimento de Campêlo *et al.* [5], além de três novos conjuntos de instâncias baseados em um método específico para o CLR.

Os resultados obtidos com os experimentos computacionais usando as instâncias do CLR e o modelo CTR mostraram que o modelo também tem um bom desempenho para essa versão do problema de recoloração convexa. Esses experimentos nos ajudaram a conhecer o problema e nos deram dicas sobre o que pode torná-lo mais difícil de resolver.

Com o primeiro experimento observamos que quando o CTR e o CLR tem o mesmo número de classes de cores, o CLR pode ser mais difícil de resolver. Trabalhos anteriores mostraram que o modelo CTR usa mais tempo para resolver instâncias do CTR que possuem mais cores. Isto era esperado, já que o problema é NP-difícil mesmo quando temos apenas dois vértices por classe de cor.

As instâncias que tinham muitas cores na versão CTR, e eram as mais difíceis de resolver, não tinham muitas cores na versão CLR. Isso teve um impacto grande no tempo de resolução dessas instâncias.

No nosso segundo experimento, buscamos reduzir variáveis que podem impactar a resolução de uma instância do CLR. Para isso, fixamos os tamanhos das árvores e de seus respectivos números de folhas. Com essas árvores, criamos dois conjuntos de instâncias com colorações diferentes. Nesse experimento, percebemos que as instâncias que usaram mais tempo para terminar de executar foram aquelas em que o número de vértices por classe de cor era menor ou igual a três. Isto é, instâncias que podem ser resolvidas em tempo polinomial. Mesmo construindo instâncias difíceis do CLR (do ponto de vista prático), com diferentes distribuições dos vértices por classes de cores, o modelo CTR continua sendo uma opção viável para resolver esse problema.

Nesta dissertação, consideramos apenas a versão uniforme do CTR. Em trabalhos futuros pretendemos investigar o problema com a versão ponderada. Outra abordagem interessante, que também pretendemos investigar é a variante do problema que considera árvores filogenéticas mais elaboradas, isto é, que representam mais de uma característica. Isso implicaria em tratar diferentes colorações sobre a mesma árvore de modo que todas sejam convexas. Neste caso, a função de custo poderia ser alterada para contabilizar quantas cores foram alteradas ou quantos vértices tiveram suas cores alteradas.

Capítulo 4

Recoloração Convexa de Grafos

Este capítulo aborda o problema de Recoloração Convexa em grafos gerais. Na seção 4.1, definimos o problema, além de apresentarmos um levantamento da literatura. Na seção 4.2, descrevemos os componentes principais do GRASP proposto nesse trabalho para o problema de recoloração convexa de grafos gerais. Na seção 4.3 apresentamos em detalhes um modelo de programação linear inteira da literatura que foi utilizado para medir a qualidade da solução da heurística. Na seção 4.4, descrevemos os experimentos realizados e discutimos seus resultados. Por último, na Seção 4.5 apresentamos as conclusões e trabalhos futuros.

4.1 Introdução

Alguns relacionamentos entre espécies não podem ser representadas com árvores filogenéticas devido à sua complexidade. Por esse motivo redes são estruturas mais adequadas para representar esses relacionamentos. Essas estrutras são chamadas de redes filogenéticas e estão propenças aos mesmos problemas que árvores filogenéticas. Desse modo, dados uma rede filogenética colorida, ainda estamos interessados em encontrar o menor número de vértices que precisam ser recoloridos de modo que a coloração dessa rede seja convexa [10].

Chamamos de Recoloração Convexa, ou *Convex Recoloring* – CR, a versão do problema de recoloração em que o tipo de grafo da entrada não é especificado. Na Definição 4.1, essa versão é apresentada formalmente. Note que, o objetivo e a saída são os mesmos das versões estudadas nos capítulos anteriores. Entretanto, nessa versão, o subgrafo induzindo pela classe de cores não precisa ser uma subárvore, basta ser conexo. Além disso, a entrada também é alterada, de modo a refletir a relaxação no tipo do grafo.

Definição 4.1 RECOLORAÇÃO CONVEXA – CR

Entrada: Um grafo $G = (V, E)$, um conjunto de cores $\mathcal{C}$, uma coloração parcial $C : V \to \mathcal{C} \cup \{\emptyset\}$ e uma função de custo w.

Objetivo: Encontrar uma coloração boa C' com custo mínimo, $\min(\sum_{v \in R_C(C')} w(v))$.

Saída: Uma coloração boa C'.

Outra aplicação para o CR é nas interações proteina-proteina. Nessa aplicação, a cor de um vértice representa uma função particular da célula. Nesse caso, é esperado que o subgrupo celular responsável por uma função forme uma única subred monocromática.

O problema CR também é provado ser NP-difícil, mesmo quando a coloração inicial é total e usa apenas duas cores [30]. Essa prova é feita através de uma redução do problema de cobertura por conjuntos. Campêlo *et al.* [6] estudaram o problema em *grids* e provaram que o mesmo é NP-difícil, usando uma redução do problema de cobertura por vértices conectados.

Campêlo *et al.* [6] introduziram uma versão relaxada do problema chamado de CRkONE onde apenas uma das k classes de cores deve ser convexa. Esse problema também é NP-difícil em *grids*. Os autores também provaram que para $k \geq 2$, tanto o CR quanto o CRkONE são inaproximáveis com um fator de $c \ln n$, onde c é uma constante qualquer, mesmo em grafos bipartidos, a menos que P=NP. Apesar dos resultados fortes sobre a complexidade do problema, Campêlo *et al.* [6] apresentaram um algoritmo polinomial para o CR com duas cores na coloração inicial e para o CRkONE específica para quando o grafo de entrada é um $(q, q-4)$–grafo com q fixo (classe que inclui os cografos e os grafos com poucos P_4s).

Kammer e Tholey [22] apresentaram uma $(2+\epsilon)$–aproximação para grafos ponderados de largura arbórea limitada. Esse algoritmo é uma extensão do trabalho de Bar-Yehuda e Feldman [2] que tratava do problema em árvores.

Campêlo *et al.* [7] propuseram um modelo de programação linear inteira para o CR, que possui um número polinomial de variáveis de decisão, mas exponencial de restrições. Já Moura [31] propôs um modelo de programação linear inteira com um número polinomial de restrições, mas exponencial de variáveis de decisão, acompanhando por um procedimento de geração de colunas apenas para o problema em árvores (CTR). Ambos os trabalhos apresentam experimentos computacionais apenas para o CTR.

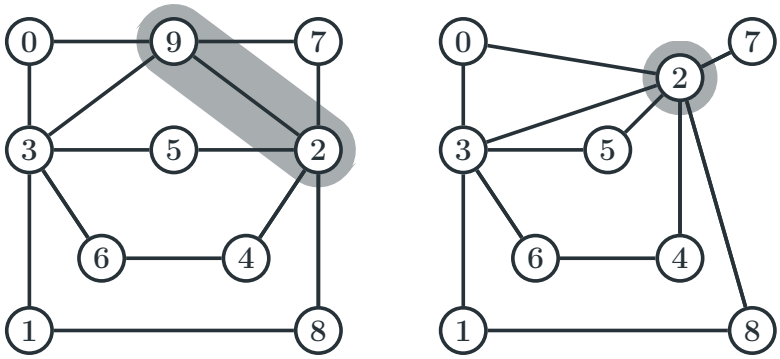
No melhor do nosso conhecimento, não existem experimentos computacionais ou abordagens heurísticas para o CR. Neste capítulo, apresentamos um heurística GRASP para o CR, além de extensivos experimentos computacionais com essa heurística e com o modelo de Campêlo *et al.* [7], que são apresentados a seguir.

4.2 *Greedy Randomized Adaptive Search Procedure*

Nesta seção, detalhamos os procedimentos de construção de uma solução e descrevemos vizinhanças para uma solução do CR que podem ser usadas na fase de busca local. Além disso, também definimos procedimentos de pós-processamento que são facilmente incorporados ao algoritmo base do GRASP, descrito no Capítulo 2 (Algoritmo 1). Os procedimentos descritos nesta seção consideram apenas a versão uniforme do CR, isto é, o custo de recolorir um vértice é igual a 1.

4.2.1 Construção de uma solução gulosa

Na Linha 4 do Algoritmo 1, o GRASP executa uma subrotina que cria uma solução gulosa aleatorizada. Nesta seção apresentamos o algoritmo para a construção dessa solução.



(a) grafo incial (b) após a operação

Figura 4.1: Ilustração do procedimento de contração. Os dois vértices destacados na Figura 4.1(a) (vértices 9 e 2) indicam onde a operação será aplicada. Na Figura 4.1(b) temos o grafo resultante da operação de contração da aresta que conecta os vértices 9 e 2.

Esse algoritmo pode ser facilmente modificado para que diferentes critérios gulosos sejam usados. Desse modo, desenvolvemos dois conjuntos de critérios que podem ser utilizados na construção.

Uma operação muto importante durante a construção de uma solução é a contração de uma aresta. Seja $e = \{u, v\}$ uma aresta de um grafo $G = (V, E)$. Uma contração da aresta e remove o vértice u e adiciona sua vizinhança na vizinhança do vértice v, tal que, $N(v) = N(v) \cup N(u) \backslash \{u\}$. Além disso, dizemos que o vértice v *representa* o vértice u que foi removido. A Figura 4.1 ilustra esse procedimento.

Considere o grafo na Figura 4.1(a), ao fazermos a contração na aresta que conecta os vértices 9 e 2 obtemos o grafo da Figura 4.1(b). Neste último, o vértice 9 foi removido e a vizinhança do vértice 2 passa a ser $\{4, 5, 7, 8, 9\} \cup \{0, 3, 7\} \backslash \{9\} = \{0, 3, 4, 5, 7, 8\}$. A seguir, detalhamos, no Algoritmo 2, a construção de uma solução que utiliza esse procedimento.

O Algoritmo 2 recebe como entrada um grafo G e sua coloração D, além do parâmetro α que determina o tamanho da lista restrita de candidatos (RCL). Primeiramente, são criadas cópias do grafo e da coloração inicial (Linhas 2 e 3). Em seguida, todas as arestas cujos vértices extremos possuem a mesma cor são contraídas (Linha 4). Isso faz com que cada componente conexa monocromática seja representada por apenas um vértice.

Os passos seguintes do algoritmo são repetidos até que não existam mais cores ruins no grafo colorido (H, D). Primeiro selecionamos dentre os vértices de H aqueles que poderão ser recoloridos na iteração atual (Linha 6). Em seguida, esses vértices selecionados são ordenados de acordo com um critério guloso e armazenados em L (Linha 7). O próximo passo é criar a RCL, que separa os $\alpha\%$ melhores vértices de L (Linha 8).

Com a RCL criada, selecionamos um vértice v aleatoriamente (Linha 9) que será recolorido (Linha 10). Com essa recoloração, espera-se que uma nova componente monocromática com tamanho maior que um tenha sido formada. Assim, podemos executar novamente uma contração das arestas que incidem sobre vértices de mesma cor (Linha 11).

Ao final das iterações teremos uma recoloração convexa D do grafo H, onde cada cor aparece no máximo em um vértice. Note que, a menos que não tenha sido feita nenhuma contração, H e G são grafos diferentes e D não é uma coloração de G. Contudo, podemos

Algoritmo 2 GRASP – *Fase de construção da solução*

```
1: função SOLVE(G, C, α)
2:     seja D uma cópia de C        ▷ coloração
3:     seja H uma cópia de G        ▷ grafo
4:     contraia todas as arestas de G com a mesma cor nos extremos
5:     enquanto existe uma cor ruim em (H, D) faça
6:         V' ← vértices de H candidatos a recoloração
7:         L ← lista de candidatos V' ordenada
8:         RCL ← porcentagem dos melhores candidatos em L
9:         seja v um vértice selecionado aleatoriamente da RCL
10:        troque a cor de v em D
11:        contraia todas as arestas de v que o conectam com um vértice de mesma cor
12:    fim enquanto
13:    C' ← transforme D em uma coloração de G
14:    devolve C'
15: fim função
```

obter uma coloração de G a partir de D, pois todos os vértices de G ou estão em H, ou são representados por um dos vértices de H.

Note que ainda não definimos nenhum critério guloso para a criação de uma solução. O Algoritmo 2 é um *framework* que pode ser usado com diferentes critérios gulosos para selecionar vértices candidatos (Linha 6), ordenar os vértices selecionados (Linha 7) e escolher a nova cor para o vértice sorteado (Linha 8). Desse modo, desenvolvemos dois conjuntos de critérios para selecionar, ordenar e recolorir um vértice. Esses conjuntos são descritos a seguir.

Conjunto de critérios *ratio*

O primeiro conjunto que desenvolvemos é chamado de *ratio*. Nesse conjunto, todo vértice de H que possui uma cor válida associada a ele é um vértice candidato a recoloração, isto é, $V' = \{v \in V(H) \mid D(v) \neq \emptyset\}$. A ordenação dos vértices candidatos é feita de acordo com dois critérios: *peso* e *fator de proporção*.

O *peso* de um vértice é dado pelo número de vértices que ele representa. Note-se que, ao fazermos uma operação de contração, o vértice que permanece no grafo é tido como representante do vértice que foi removido. Além disso, o *peso* de um vértice aqui não é o mesmo que o custo de recolorir um vértice na versão não uniforme do problema de recoloração.

Já o *fator de proporção* de um vértice é dado pela frequência da cor mais comum na vizinhança de um vértice dividido pelo tamanho da vizinhança. Os candidatos são ordenados em ordem crescente de *peso* com os empates quebrados pelos valores de *fator de proporção* em ordem decrescente. Quando recolorimos um vértice que possui um *peso* maior que um, recolorimos também os vértices que ele representa. Ao darmos preferência aos vértices com menor *peso*, buscamos fazer uma recoloração que pode custar menos. Ao quebrarmos os empates com *fator de proporção*, damos preferência aos vértices que unem mais componentes monocromáticas e podem ter o menor custo.

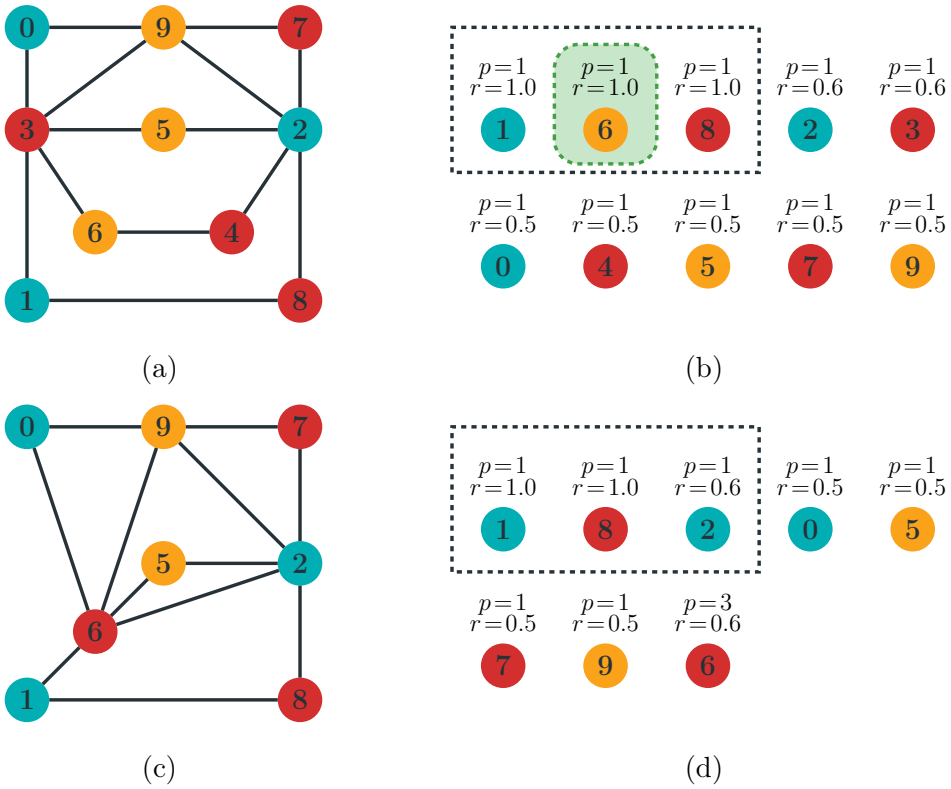


Figura 4.2: Construindo uma solução com o conjunto *ratio*. Usando o grafo colorido da Figura 4.2(a) como entrada, os vértices candidatos são ordenados como é mostrado na Figura 4.2(b). Supondo que $\alpha = 0.3$ e que o vértice 6 foi recolorido, o grafo da Figura 4.2(c) resulta na contração das arestas que tem a mesma cor nas pontas. Na Figura 4.2(d) é apresentada a nova ordenação dos candidatos.

O último critério a ser definido neste conjunto é o de recoloração do vértice v sorteado, que é definido de acordo com os casos:

- Se a frequência da cor mais comum na vizinhança de v é maior que 1, então v recebe essa cor;
- Se não existem pelo menos dois vértices com a mesma cor na vizinhança de v, mas existe um vértice na vizinhança que possui uma cor convexa, então v recebe essa cor convexa;
- Se v não se encaixa nos casos anteriores, então v tem sua cor removida.

No primeiro caso, ao recolorir o vértice com a cor mais frequente, e que aparece mais de uma vez na vizinhança, criamos uma componente monocromática de tamanho maior que 2, ou seja, unimos duas componentes que tinham a mesma cor, esse é o caso ideal. Quando isso não acontece, podemos ou usar uma cor convexa da vizinhança, ou remover a cor do vértice. Nesses casos queremos remover v da lista de candidatos sem afetar os vértices da vizinhança que poderiam ser usados para conectar outras componentes de mesma cor. Na Figura 4.2 exemplificamos a construção de uma solução usando os critérios do conjunto *ratio*.

Começando o processo com a grafo colorido da Figura 4.2(a), não são feitas contrações iniciais, pois, não existem arestas com as duas extremidades da mesma cor na entrada. Como todos os vértices possuem uma cor válida, todos são candidatos a recoloração. Em seguida devemos calcular os critérios de ordenação para cada vértice. Como nenhuma contração foi feita ainda, todos os vértices têm *peso* 1.

O *fator de proporção* é calculado para cada vértice baseado na mesma topologia. Por exemplo, o vértice 3 possui uma vizinhança de tamanho 5 e a cor mais comum é o amarelo (●), que aparece três vezes. Logo, o *fator de proporção* do vértice 3 é $3/5 = 0.6$. A Figura 4.2(b) mostra o resultado da ordenação dos vértices candidatos pelos seus respectivos valores de *peso* (p) e *fator de proporção* (r).

Considerando $\alpha = 0.3$, um retângulo tracejado demarca na Figura 4.2(b) quais os vértices que compõem a RCL. Suponha que o vértice 6, demarcado com um retângulo verde na mesma figura, foi sorteado para ser recolorido. Como a cor mais frequente na sua vizinhança é a vermelha (●), e essa cor aparece mais de uma vez, o vértice 6 é recolorido com a cor vermelha.

Depois da recoloração, as arestas $\{3, 6\}$ e $\{4, 6\}$ possuem as duas extremidades com cores iguais, logo devem ser contraídas. O resultado dessa contração é mostrado na Figura 4.2(c). Esse passo conclui uma iteração das Linhas 5–12 do Algoritmo 2, e como ainda existem cores ruins no grafo colorido, os passos acima devem ser repetidos. A Figura 4.2(d) mostra como seria a seleção e ordenação dos vértices candidatos na iteração seguinte. Note que contração dos vértices na iteração anterior afetou tanto o *peso* quanto o *fator de proporção* do vértice 6.

Conjunto de critérios *union*

O segundo conjunto de critérios que desenvolvemos para criar uma solução gulosa é chamado de *union*. Nesse critério, um vértice é considerado candidato se sua cor não é convexa ou se possui pelo menos um vizinho cuja cor não é convexa. Novamente, o ordenação usa duas características dos vértices, nesse caso os valores de *custo real* e *fator de união*.

O *custo real* de um vértice v é dado pelo número de vértices que v representa e que ainda não foram recoloridos anteriormente. Desta forma, contabilizamos o custo exato de recolorir o vértice naquele momento do processo de criação de uma solução.

Já o *fator de união* do vértice v é o maior número de vértices que podem ser adicionados a maior componente de cor c na vizinhança de v, se o vértice v for recolorido com a cor c dentre todas as cores c dos vértices vizinhos de v. Esse valor é computado da seguinte maneira. O fator da cor c relativo ao vértice v é igual à soma dos pesos dos vizinhos de v coloridos com c menos o maior deles. O *fator de união* do vértice é o maior dos fatores das cores de sua vizinhança.

Os vértices candidatos são ordenados em ordem crescente pelo valor de *custo real* com empates quebrados pelos valores de *fator de união* em ordem decrescente. Diferente do critério *ratio*, aqui consideramos o custo real de recoloração e não apenas uma estimativa do custo de recoloração. O desempate no *union* considera também o peso dos vértices, que pode ser visto como o tamanho real das componentes conexas que serão unidas caso o

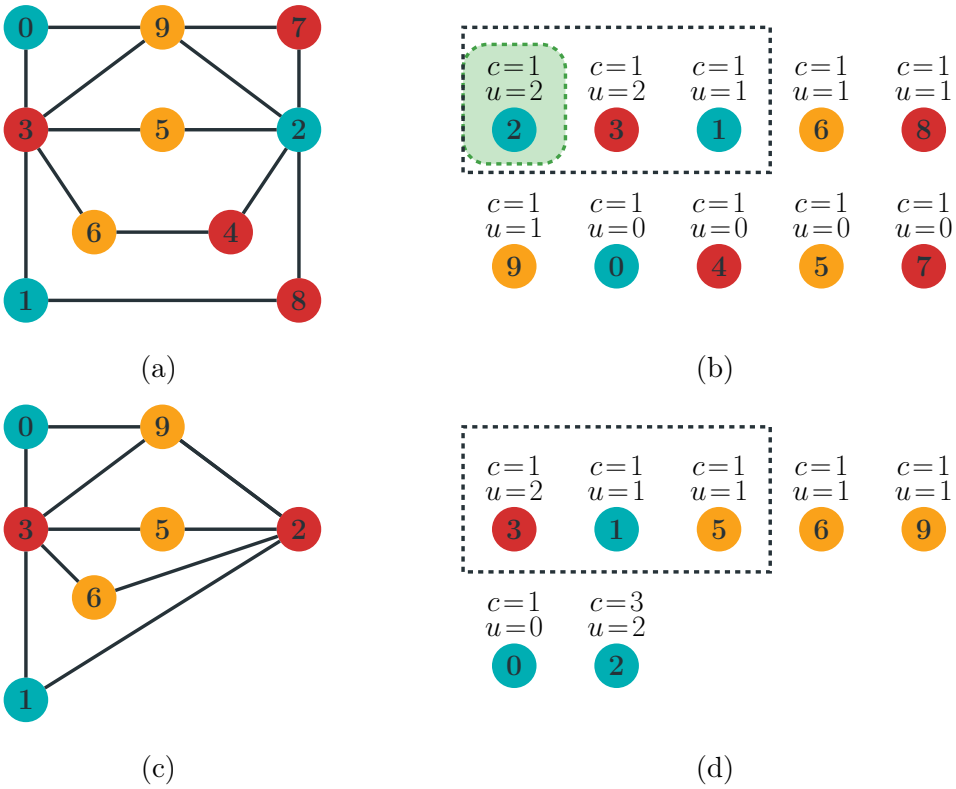


Figura 4.3: Construindo uma solução com o conjunto *union*. Usando o grafo colorido da Figura 4.3(a) como entrada, os vértices candidatos são ordenados como é mostrado na Figura 4.3(b). Supondo que $\alpha = 0.3$ e que o vértice 2 foi recolorido, o grafo da Figura 4.3(c) resulta da contração das arestas de mesma cor. Na Figura 4.3(d) é apresentada a nova ordenação dos candidatos.

vértice candidato seja recolorido. De modo que, no *union*, damos preferência aos vértices que unem as maiores componentes monocromáticas e que têm o menor custo naquele momento da construção da solução.

Além disso, a nova cor do vértice, caso ele seja sorteado para ser recolorido, é a cor referente ao maior fator do vértice. Na Figura 4.3 exemplificamos a construção de uma solução usando os critérios do conjunto *union*.

Tendo o grafo colorido da Figura 4.3(a) como entrada, todos os vértices são selecionados como candidatos, já que todas as cores são ruins. Como todo vértice representa apenas ele mesmo e não houve recolorações, o *custo real* de recoloração para todos eles é 1. O *fator de união* é calculado como descrito acima. Para exemplificar esse cálculo consideremos o vértice 2. As cores vermelha (●) e amarela (●) aparecem na vizinhança do vértice 2, de modo que precisamos calcular o fator apenas para essas duas cores.

Os dois vértices amarelos na vizinhança do vértice 2 possuem peso 1. Logo, se recolorirmos o vértice 2 com a cor amarela, adicionaremos apenas $1+1-1 = 1$ vértice na maior delas (além do vértice 2), que é o fator da cor amarela para esse vértice. Já os vértices vermelhos da vizinhança do vértice 2 também possuem peso 1, mas temos 3 componentes, de modo que, o fator dessa cor é $1+1+1-1 = 2$. Logo, o *fator de união* do vértice 2 é 2.

Os valores de *custo real* (c) e *fator de união* (u) para todos os vértices candidatos

são apresentados acima de cada vértice na Figura 4.3(b). Note que os vértices já estão ordenados. Considere novamente que $\alpha = 0.3$, os vértices candidatos que compõem a RCL estão demarcados com um retângulo tracejado como no exemplo anterior.

Suponha que o vértice 2 foi sorteado para ser recolorido. Como a cor vermelha tem maior *fator de união*, o vértice 2 recebe esta cor. A Figura 4.3(c) mostra o grafo após a contração das arestas $\{2,4\}$, $\{2,7\}$ e $\{2,8\}$, tinham vértices de mesma cor nas extremidades após a recoloração do vértice 2.

Na iteração seguinte da fase de construção, teremos novamente todos os vértices como candidatos, como mostra a Figura 4.3(d). Essa figura também mostra como os vértices seriam ordenados. Note que, apesar do vértice 2 representar os vértices $2, 4, 7$ e 8 e ter peso 4, o custo de recoloração é apenas 3, pois o vértice 2 já foi recolorido.

4.2.2 Vizinhanças para a busca local

Nesta seção definimos três vizinhanças para uma solução do problema de recoloração convexa. São elas: *vizinhança simples*, *vizinhança estendida* e *vizinhança de troca*. Estas vizinhanças são usadas no procedimento da Linha 5 do Algoritmo 1 (Capítulo 2). A ideia básica é desfazer as recolorações de vértices que podem ser removidos de suas classes de cor atuais. Chamamos de *reverter* um vértice a operação de que desfaz a recoloração de um vértice v, isto é, a reversão de v o retorna à sua classe de cor original. A seguir detalhamos essas três vizinhanças.

Vizinhança Simples

Dados uma recoloração convexa C' de um grafo colorido (G, C), a *vizinhança simples* procura por um vértice que foi recolorido e que não é um ponto de articulação na sua classe atual, isto é, pode ser removido da classe e ela permanecerá conexa.

Seja v um vértice tal que $C'(v) = c$, $C(v) = d$ e $c \neq d$ e que não é um ponto de articulação. Seja u um vértice na vizinhança de v tal que $C'(u) = d$. Caso u exista podemos reverter v para a sua cor original, garantindo que as duas classes de cor c e d são convexas após a reversão. O procedimento de busca local é executado enquanto houverem vértices que atendam aos critérios acima. A Figura 4.4 mostra um exemplo de execução da busca local com a *vizinhança simples*.

Suponha que o grafo colorido da Figura 4.4(a) é uma recoloração convexa do grafo colorido da Figura 4.3(a). Os vértices $\{1, 2, 3, 7, 9\}$ foram recoloridos e suas cores originais estão demarcadas pelos círculos externos em torno de cada vértice. Por exemplo, o vértice 2 era colorido originalmente com azul (●), mas na recoloração recebeu amarelo (●).

Note que o vértice 2 pode ser removido da classe de cor amarelo sem que esta deixe de ser conexa, como mostrado na Figura 4.4(b). Além disso, o vértice 2 é vizinho de pelo menos um vértice que está colorido com azul, de modo que podemos revertê-lo. Como não existem outros vértices que podem ter sua recoloração revertida, a busca local encerra e retorna uma coloração que custa uma unidade a menos, apresentada na Figura 4.4(c).

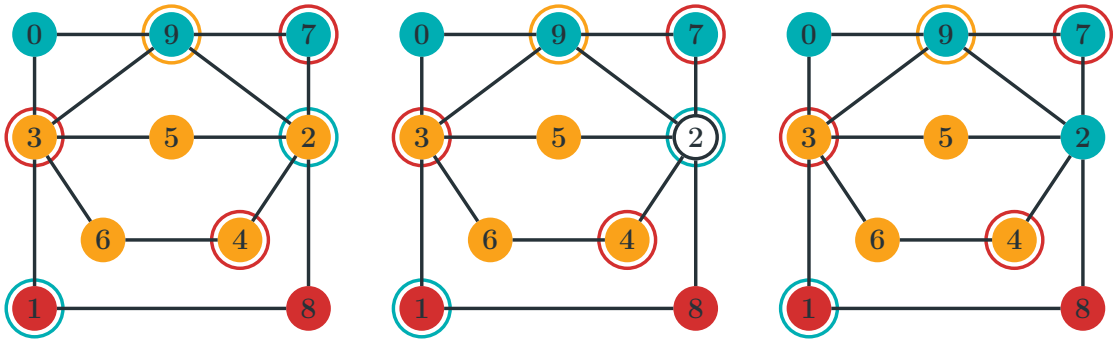


Figura 4.4: Busca local com a *vizinhança simples*. Na Figura 4.4(a), temos uma recoloração do grafo colorido da Figura 4.3(a). O vértice 2 pode ser removido da sua classe de cor atual, como na Figura 4.4(b), e revertido para sua classe original (Figura 4.4(c)).

Vizinhança Estendida

A *vizinhança estendida* incrementa a ideia da *vizinhança simples*, permitindo uma distância maior entre um vértice recolorido que não é ponto de articulação em sua classe de cor original, sendo que na *vizinhança simples* essa distância é fixada em 1.

Dados uma recoloração convexa C' de um grafo colorido (G, C), construímos um conjunto $\mathcal{S}$ de vértices que foram descoloridos em C' e o incrementamos da seguinte maneira: encontre um vértice v que não é ponto de articulação, remova sua cor, ou seja, faça com que $C'(v) = \emptyset$, e o adicione a $\mathcal{S}$. Repita esses passos até que o conjunto não possa mais ser incrementado. Note que podemos precisar de algumas iterações para que o conjunto $\mathcal{S}$ fique completo, pois a remoção de um vértice de uma classe de cor pode tanto criar quanto remover pontos de articulação na classe.

Com o conjunto $\mathcal{S}$ criado, temos uma recoloração que ainda é convexa, pois não removemos as cores de pontos de articulação, e que tem o mesmo custo da recoloração inicial. No próximo passo da *vizinhança estendida*, para cada vértice $v \in \mathcal{S}$ procuramos o menor caminho de v até um vértice w tal que $C'(w) = C(v)$, com a restrição de que os vértices internos desse caminho são vértices de $\mathcal{S}$. Seja x o vértice de $\mathcal{S}$ que possui o menor caminho até a sua classe de cor original, removemos x e os vértices do seu respectivo caminho de $\mathcal{S}$ e os colorimos com a cor original de x. Com isso conseguimos reverter o vértice x.

O cálculo das distâncias e a reversão dos vértices são repetidos até que não existam vértices em $\mathcal{S}$ que possam ser conectados às suas classes originais, usando apenas vértices descoloridos. A Figura 4.5 mostra um exemplo de execução da busca local com a *vizinhança estendida*.

A recoloração da Figura 4.5(a) é a mesma do exemplo anterior (Figura 4.4(a)). Como todos os vértices possuem uma cor nessa recoloração, começamos com o conjunto $\mathcal{S}$ vazio. A Figura 4.5(b) mostra uma possível formação do conjunto $\mathcal{S}$, onde um vértice de cor branca significa que ele não tem uma cor associada a ele. Nessa figura, a distância até um vértice de sua cor original está indicada acima de cada vértice que foi descolorido. Note que o vértice 1 não pode ser conectado com um vértice da cor azul (●) usando apenas

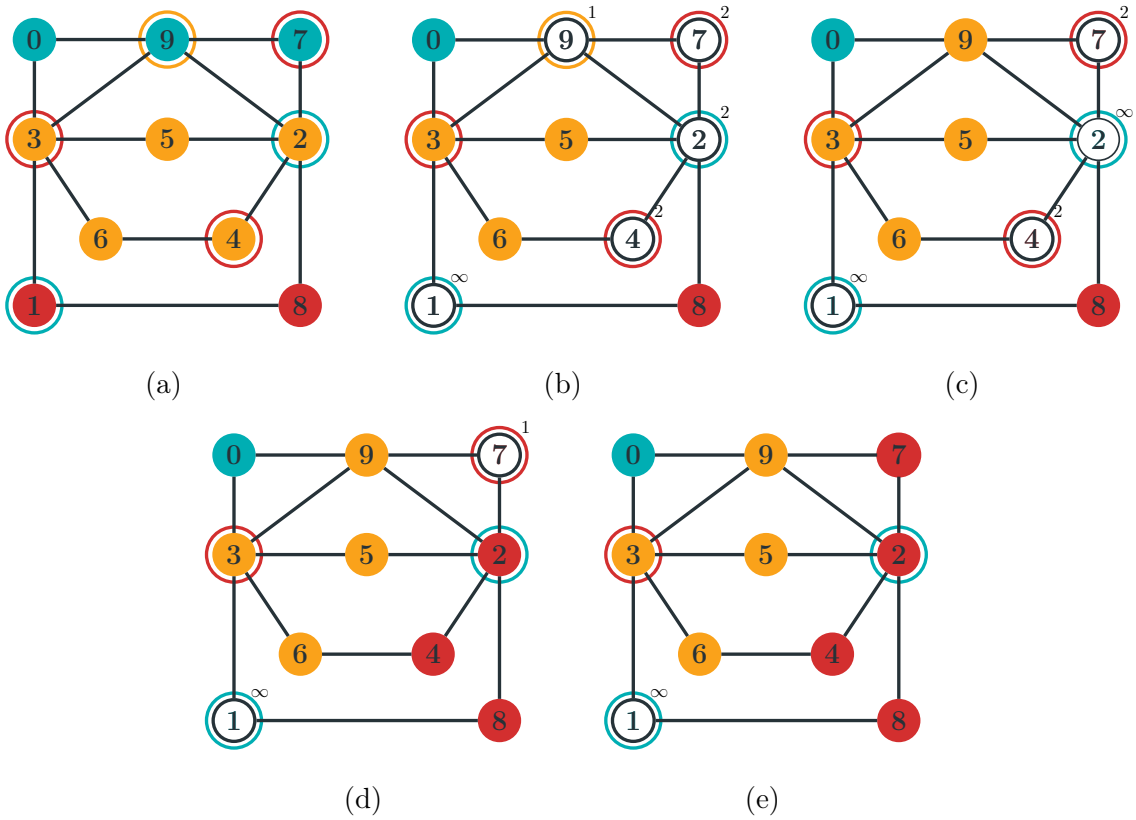


Figura 4.5: Busca Local com a *vizinhança estendida*. Considerando a recoloração da Figura 4.5(a), o procedimento remove a cor dos vértices que não são pontos de articulação (Figura 4.5(b)) e calcula as distâncias dos vértices descoloridos até as suas classes. Nas Figuras 4.5(c)–4.5(e) os vértices com a menor distância são revertidos e as distâncias recalculadas.

vértices descoloridos, por isso sua distância é marcada como ∞.

Como o vértice mais próximo de sua cor original é o vértice 9, este tem sua cor revertida e as distâncias dos vértices restantes são atualizadas (Figura 4.5(c)). Note que agora o vértice 2 não possui caminho descolorido até um vértice azul. As figuras seguintes, Figuras 4.5(d) e 4.5(e) mostram as próximas reversões, onde os vértices 4 e 7 voltam para suas classes de cor originais. A busca local termina a execução com uma recoloração que custa três unidades a menos.

Vizinhança de Troca

A última vizinhança que definimos é chamada de *vizinhança de troca*. Dados um grafo colorido (G, C) e uma de suas possíveis recolorações convexas C', construímos o conjunto $\mathcal{S}$ dos vértices que podem ser removidos da classe de cor a qual pertencem na recoloração C' sem desconectá-la, definida como na *vizinhança estendida*.

O primeiro passo dessa vizinhança verifica se alguma cor desapareceu durante a recoloração. Seja d uma cor que está presente em C, mas não em C', e seja x um vértice do conjunto $\mathcal{S}$ tal que $C(x) = d$. Como $x \in \mathcal{S}$, x é revertido para sua cor original e a classe de cor d que antes era vazia em C' agora tem um único vértice x, e portanto, é convexa.

Caso x não seja único, escolhemos o vértice com o maior grau. Fazemos essa escolha para que tenhamos uma possibilidade maior de recuperar outros vértices dessa classe de cor. Ao final, x é removido do conjunto $\mathcal{S}$.

Em seguida, verificamos para cada $u \in \mathcal{S}$ se existe um caminho (u, v, w) tal que w está colorido com a cor original de u, ou seja, $C'(w) = C(u) = c$ e v não é um ponto de articulação em G. Note que v não precisa ser um vértice descolorido. Se o caminho descrito acima existe, então atribuímos a cor c para os vértices u e v e removemos u do conjunto $\mathcal{S}$.

O vértice v ou tem uma cor, ou faz parte de $\mathcal{S}$. Caso $v \in \mathcal{S}$, então temos uma nova recoloração convexa com custo reduzido em uma unidade, e também removemos v de $\mathcal{S}$. Caso v não faça parte do conjunto $\mathcal{S}$, ele é um vértice que não foi recolorido, ou seja, $C'(v) = C(v)$. Neste caso, ao colorir u e v com a cor c, estamos mantendo o custo de recoloração, pois, estamos apenas trocando qual vértice foi recolorido nessa solução. Esse passo da busca local termina quando não existirem mais os caminhos de tamanho 2 conectando um vértice descolorido com um vértice de sua cor original.

O último passo dessa vizinhança, verifica se existem vértices em $\mathcal{S}$ que são vizinhos de um vértice com sua cor original, e que, portanto podem ser revertidos. Esse passo é uma execução da busca local com a *vizinhança simples*. A Figura 4.6 mostra um exemplo de execução da busca local com a *vizinhança de troca*.

Suponha que o grafo colorido da Figura 4.6(a) é uma recoloração convexa do grafo da Figura 4.3(a). Nessa recoloração, trocamos a cor de 5 vértices ($\{0, 1, 2, 5, 6\}$). Note que a cor azul (●) desapareceu dessa coloração. Dada essa recoloração, o primeiro passo da busca local com a *vizinhança de troca* é construir o conjunto $\mathcal{S}$. Na Figura 4.6(b) temos uma possível formação do conjunto $\mathcal{S}$ contendo os vértices $\{0, 1, 5\}$.

O próximo passo é verificar se existem classes de cor que foram removidas da coloração. Nesse exemplo apenas a cor azul foi removida. Em seguida, procuramos por vértices que tinham a cor removida (azul) e estão no conjunto $\mathcal{S}$. Temos duas possibilidades: os vértices 0 e 1. Como ambos têm o mesmo grau, suponha que escolhemos o vértice 0 para ter sua cor revertida, como mostra a Figura 4.6(c).

Suponha agora que o vértice 5 é analisado, o caminho $(5, 3, 9)$ satisfaz às condições impostas na descrição da *vizinhança de troca*. De modo que podemos reverter a cor do vértice 5 e recolorir o vértice 3 com amarelo (●). O resultado desse passo é apresentado na Figura 4.6(d). Note que o custo da recoloração não foi reduzido.

Como não existem outros vértices descoloridos que atendem às restrições da vizinhança, continuamos para o próximo passo, que é uma execução da busca local com a *vizinhança simples*. Esse passo consegue reverter a cor do vértice 6, já que agora o mesmo é vizinho de um vértice com a sua cor original. Com esse último passo a busca local termina com a coloração da Figura 4.6(e) que custa duas unidades a menos que a recoloração inicial.

4.2.3 Pós-processamento

Nesta seção descrevemos dois pós-processamentos desenvolvidos para uso juntamente com a heurística proposta neste trabalho. Esses procedimentos são usados em fases diferentes

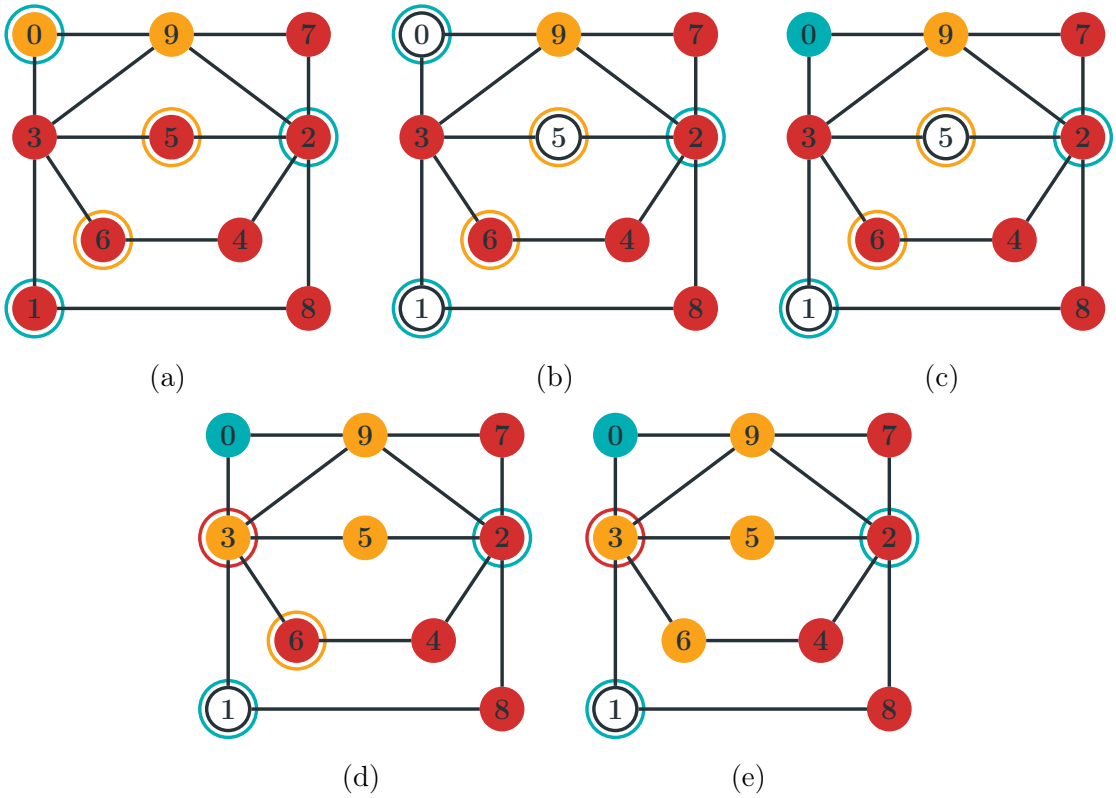


Figura 4.6: Busca Local com a *vizinhança de troca*. Suponha que o grafo da Figura 4.6(a) seja uma recoloração do grafo da Figura 4.3(a). As Figura 4.6(b) – 4.6(e) mostram os passos da *vizinhança de troca*, em ordem: descoloração dos vértices recoloridos; retorno de uma cor que sumiu; troca do vértice recolorido; execução da busca local com a *vizinhança simples*.

do GRASP, um deles é usado ao final de cada iteração e o outro é usado apenas depois que o GRASP termina. Ambos são descritos a seguir.

Mutações

O primeiro procedimento é chamado de *mutação*. Uma mutação é um conceito de algoritmos genéticos, usado aqui para designar uma alteração na solução. Essa alteração também pode ser vista como uma perturbação durante o processo de busca local.

Feo *et al.* [17] foram um dos primeiros a combinar essa ideia de perturbação na busca local com o GRASP. Em outro trabalho, Feo e Resende [15] estruturaram o processo que foi então chamado de mutação, onde a mutação é aplicada após cada execução da busca local. A *mutação* simples que desenvolvemos para o nosso problema consiste em manter a cor de apenas um vértice de uma classe, removendo a cor dos demais.

Seja C' uma recoloração e seja V_c o conjunto formado por todos os vértices com a cor c na recoloração C'. Seja x um vértice de V_c tal que x tem o maior grau, dando preferência para um x que não foi recolorido em C'. Uma *mutação* seleciona uma cor c e descolore todos os vértices do conjunto V_c com exceção de x, ou seja, $C'(v)$ para todo $v \in V_c/\{x\}$.

Após a aplicação da *mutação*, executamos uma busca local para tentar devolver a cor dos vértices que foram descoloridos. Note que ao removermos os vértices de uma classe

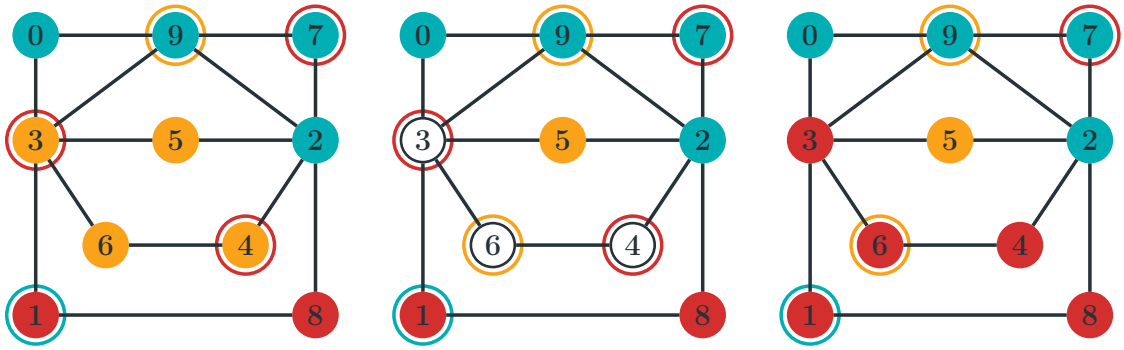


Figura 4.7: Pós-processamento com a *mutação*. Supondo que o grafo colorido da Figura 4.7(a) é uma recoloração obtida após execução da fase de busca local. Na Figura 4.7(b), temos uma possível *mutação* aplicada na solução anterior. Em seguida, outra execução da busca local é feita e a solução apresentada na Figura 4.7(c) é devolvida.

de cor, garantimos que a coloração resultante ainda é convexa. Contudo, ela pode ter um custo maior. Além disso, não temos garantia de que a solução resultante será melhor que solução inicial. Na Figura 4.7, apresentamos um exemplo de execução da *mutação*.

Suponha que o grafo colorido da Figura 4.7(a) é obtido após uma execução completa da busca local. Suponha ainda a cor amarela (●) foi selecionada para a *mutação*. O grafo colorido da Figura 4.7(b) mostra o resultado da perturbação na solução, onde apenas o vértice 5 permaneceu na classe de cor amarela. Note que a coloração ainda é válida, mas possui um custo maior que o anterior. A Figura 4.7(c) mostra uma possível solução que pode ser obtida com a execução de uma busca local no grafo com a *vizinhança estendida*. O procedimento de *mutação* termina com uma recoloração que custa uma unidade a menos.

Elite

O segundo pós-processamento é usado apenas ao final de todas as iterações do GRASP e é baseado em *Path Relinking*. Nele, usamos o conceito de soluções *elite*, isto é, um subconjunto das soluções disponíveis que têm os melhores resultados.

A ideia básica do *Path Relinking* é explorar uma sequência de operações que transforma uma solução em outra. Essa exploração é feita em busca de soluções melhores e que tenham características em comum com as duas soluções *elite* usadas [33]. Nosso pós-processamento parte dessa ideia, mas ao invés de procurar a melhor solução em um dos possíveis caminhos entre duas soluções *elite*, procuramos pela melhor solução em qualquer caminho entre as duas soluções. Esse procedimento é detalhado na sequência.

Dadas duas recolorações convexas C_1 e C_2 de um grafo G, definimos o conjunto I^{C_1,C_2} a interseção dessas duas soluções. A interseção de duas soluções é formada pelos vértices que têm a mesma cor em C_1 e C_2. Seja m o tamanho do conjunto de soluções elite, obtidas durante as k iterações do GRASP, tal que $m \leq k$. Para cada par de soluções elite (C_i, C_j), com $i < j \leq m$, criamos uma coloração parcial tal que um vértice v tem uma cor associada a ele, se e somente se, $v \in I^{C_i,C_j}$. Note que essa coloração não

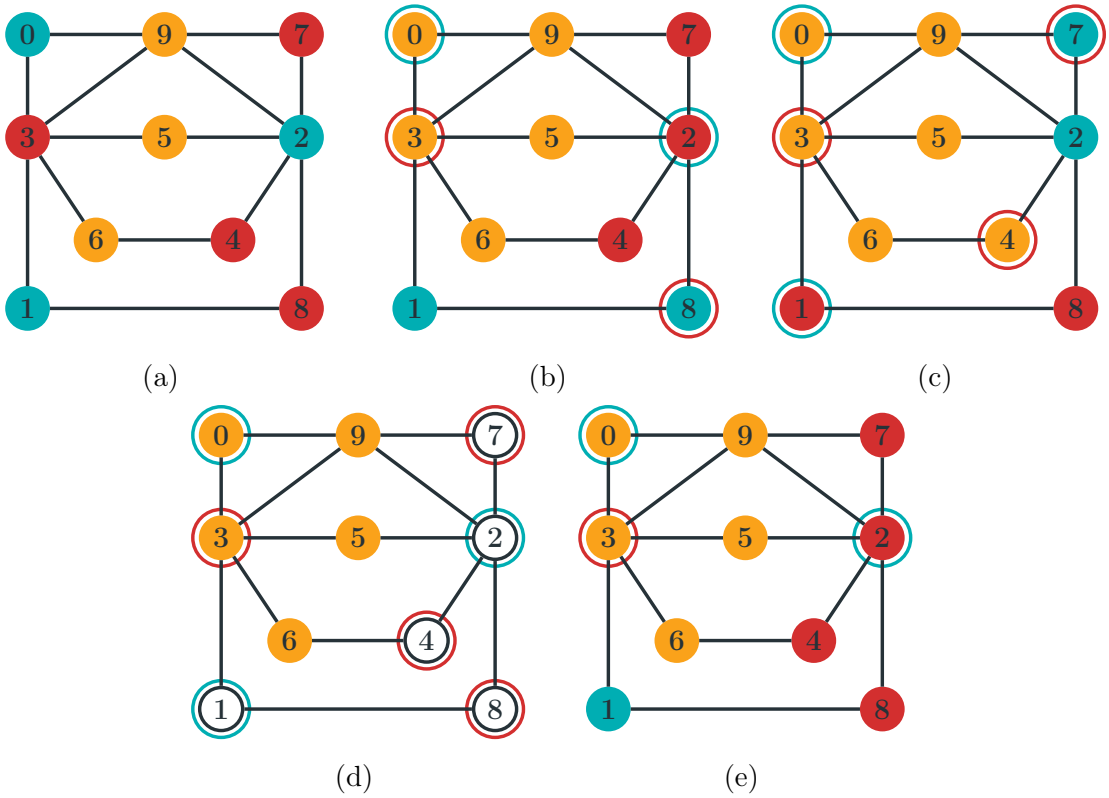


Figura 4.8: Pós-processamento com soluções *elite*. Tendo como entrada o grafo colorido na Figura 4.8(a), os grafos das Figuras 4.8(b) e 4.8(c) são possíveis recolorações convexas. A interseção dessas duas recolorações é apresentada na Figura 4.8(d). A melhor recoloração possível com os vértices da interseção fixados é dada na Figura 4.8(e).

necessariamente será convexa.

A partir da coloração obtida com a interseção, construímos uma nova recoloração convexa C' do grafo G onde os vértices de I^{C_i,C_j} não podem mudar. Como C_i e C_j são recolorações convexas de G, sabemos que C' existe. Uma forma de encontrar uma tal coloração C' de menor custo é usar um modelo de Programação Linear Inteira (PLI), como o modelo que será descrito na próxima seção, com restrições adicionais que fixam as cores dos vértices da interseção I^{C_i,C_j}.

Entretanto, para este método ser viável, o número de vértices com cor fixada deve ser grande. Além disso, ao usarmos um conjunto de soluções elite de tamanho m, teremos que resolver $\binom{m}{2}$ modelos PLI. A Figura 4.8 mostra um exemplo desse pós-processamento.

Suponha que as duas colorações nas Figuras 4.8(b) e 4.8(c) são recolorações do grafo colorido da Figura 4.8(a) com custos 4 e 5, respectivamente. A interseção das duas soluções é composta pelo conjunto $\{0, 3, 5, 6, 9\}$, e a coloração criada a partir dessa interseção é apresentada na Figura 4.8(d). Por último, a Figura 4.8(e) mostra a melhor recoloração possível que mantém a cor dos vértices da interseção. Note que essa nova recoloração tem o custo reduzido em uma unidade em relação à melhor das recolorações usadas como elite (recoloração na Figura 4.8(b)).

4.3 Modelo de Programação Linear Inteira

Nesta seção apresentamos o modelo de Programação Linear Inteira que será usado para medir a qualidade da solução dada pelas abordagens heurísticas apresentadas na seção anterior.

Existem na literatura dois modelos de Programação Linear Inteira para o CR, um proposto por Campêlo *et al.* [7] e outro proposto por Moura [31]. Foram feitos experimentos computacionais com ambos os modelos, contudo, eles tinham foco apenas em árvores filogenéticas. Além disso, o subproblema de *pricing* apresentado por Moura [31] não trata do caso de grafos gerais, por isso escolhemos a formulação de Campêlo *et al.* [7]. Além dos experimentos computacionais com árvores, Campêlo *et al.* [7] também apresentaram provas de que a formulação proposta está correta para o CR.

A formulação de Campêlo *et al.* [7] é dada pelas Equações (4.1) – (4.4). Nessa formulação, a constante $w_{v,c}$ é igual ao custo de recolorir o vértice v, dado por $w(v)$, se e somente se, o vértice v foi inicialmente recolorido com a cor c, vale lembrar que na versão uniforme $w(v) = 1$, para todo v. Caso contrário, $w_{v,c}$ é igual a 0. A variável de decisão binária $x_{v,c}$ é igual a 1 se, e somente se, o vértice v for colorido com a cor c, e igual a 0 caso contrário.

Para todo par de vértices não adjacentes (u, v), um corte por vértices Z é um conjunto de vértices que, quando removidos de G deixam u e v em componentes distintas. Assim, definimos $\Gamma(u, v)$ como o conjunto de todos os cortes por vértices minimais que separam u e v. O modelo é apresentado a seguir.

$$\max \quad \sum_{v \in V} \sum_{c \in \mathcal{C}} w_{v,c} x_{v,c} \tag{4.1}$$

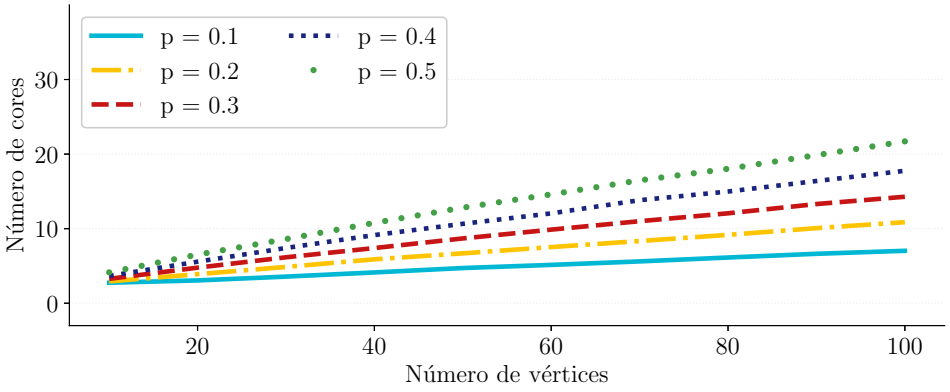
$$s.t. \quad \sum_{c \in \mathcal{C}} x_{v,c} \leq 1 \qquad \forall v \in V \tag{4.2}$$

$$x_{u,c} + x_{v,c} - \sum_{z \in Z} x_{z,c} \leq 1 \quad \forall \{u, v\} \notin E,\ Z \in \Gamma(u, v),\ c \in \mathcal{C} \tag{4.3}$$

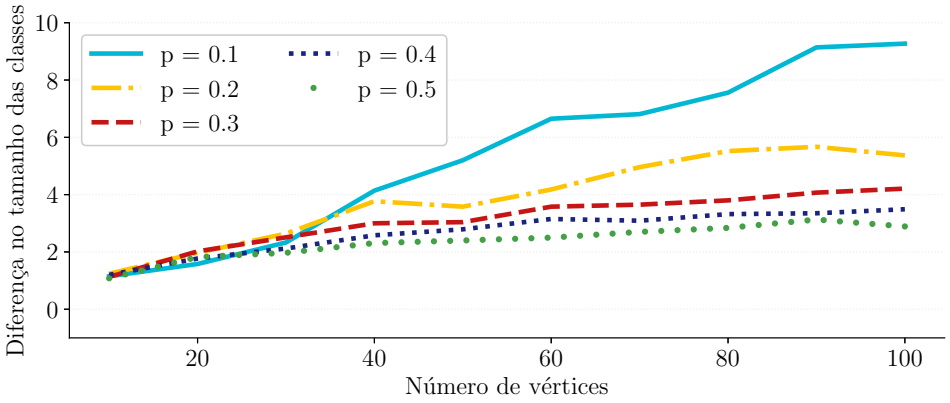
$$x_{v,c} \in \{0, 1\} \qquad \forall v \in V,\ \forall c \in \mathcal{C} \tag{4.4}$$

A Equação (4.1) define a função objetivo do modelo. Essa função maximiza o número de vértices que mantém sua cor na recoloração. Note que o problema é geralmente tratado como uma minimização do número de vértices recoloridos. Contudo, maximizar o número de vértices que mantêm suas cores iniciais é equivalente.

As Inequações (4.2) restringem que um vértice receba no máximo uma cor. Desse modo, a modelagem admite uma solução que pode conter vértices descoloridos. Já as Inequações (4.3) garantem que todo par de vértice não adjacentes que possuem a mesma cor estão conectados. Por último, as Inequações 4.4 restringem o domínio das variáveis de decisão $x_{v,c}$.



(a) número de cores



(b) diferença no tamanho das classes

Figura 4.9: Características das colorações iniciais. No primeiro gráfico (Figura 4.9(a)) apresentamos o número médio de cores das instâncias. No segundo gráfico (Figura 4.9(b)) apresentamos média da diferença de tamanho entre a maior e a menor classe de cor.

4.4 Experimentos computacionais

Os experimentos reportados nesta seção foram executados em uma máquina com processador Intel^{R} Core^{TM} i7 3.40GHz, 32GB de memória RAM e sistema operacional Ubuntu 18.04 LTS. Apresentamos resultados tanto para o modelo PLI quanto para o GRASP, que foram implementados usando a linguagem de programação `C++` e compilados usando `g++` (versão 5.4), com as *flags* `C++11` e `-O3`. Usamos o resolvedor de programação linear inteira `IBM CPLEX` na versão 12.8, com as funções de *presolve* e de paralelismo do `CPLEX` desativadas.

Ao melhor do nosso conhecimento, não existem experimentos computacionais para o CR reportados na literatura. Por esse motivo desenvolvemos um *benchmark*[1] de instâncias para o problema de recoloração convexa em grafos gerais. Ao criarmos esse *benchmark*, buscamos instâncias que possam ser consideradas difíceis.

Usamos o termo coloração própria para designar uma coloração onde dois vértices adjacentes não podem ter a mesma cor. Deste modo, todas as classes de cor de uma coloração própria são ruins, considerando as definições de coloração convexa. Durante as análises feitas no Capítulo 3, percebemos que não só o número de cores, como também o número de cores ruins influenciam na dificuldade de uma instância. Por isso, decidimos

[1]Disponível em `www.loco.ic.unicamp.br/files/instances/convex-recoloring`

criar instâncias cuja coloração inicial é uma coloração própria.

Um grafo aleatório produzido usando o procedimento de Erdös-Rényi precisa dos parâmetros n e p, onde n indica o número de vértices e p a probabilidade de adicionar uma aresta no grafo. Com esse procedimento, criamos 100 grafos conexos[2] para cada par (n, p) com $n \in \{10, 20, \ldots, 100\}$ e $p \in \{0.1, 0.2, \ldots, 0.5\}$. No total foram gerados 5000 grafos de tamanhos e densidades variados.

Para cada um dos grafos descritos acima, criamos uma coloração própria usando um algoritmo guloso básico [4]. Esse algoritmo usa uma lista dos vértices ordenados pelo grau. O primeiro passo atribui a cor com o menor valor de representação disponível ao vértice de maior grau, adicionando cores novas quando não podemos usar uma cor que já existe. Em seguida o vértice que foi colorido é removido da lista, e o passo anterior é repetido até que a lista esteja vazia.

Após a criação da coloração própria, aplicamos um balanceamento nas classes de cores, de modo que seus tamanhos fossem mais equilibrados. Nesse balanceamento, retiramos vértices das maiores classes e os adicionamos, quando possível, nas classes menores mantendo as características de uma coloração própria. Com esse procedimento, conseguimos impedir que classes de cor tenham tamanho 1, que são trivialmente convexas. Assim, em todas as instâncias, cada classe de cor induz um subgrafo desconexo.

Na Figura 4.9 temos dois gráficos que representam características das instâncias criadas para o *benchmark*. Em ambos os gráficos, as instâncias foram separadas pelo valor de p e o eixo x indica o número de vértices da instância. Na Figura 4.9(a) temos um gráfico cujo eixo y indica a média do número de cores. Vemos que, ao aumentarmos os valor p, o número de cores também aumenta e, por consequência, o número de vértices por classe de cor diminui.

No gráfico da Figura 4.9(b), mostramos a média da diferença, em unidades, entre a maior e a menor classe de cor. Esse gráfico mostra que as maiores diferenças estão nas instâncias $p = 0.1$, que possuem os menores números de cores. Isso também indica que o procedimento de balanceamento não foi tão eficaz, apesar de ter sido capaz de evitar as classes de tamanho 1.

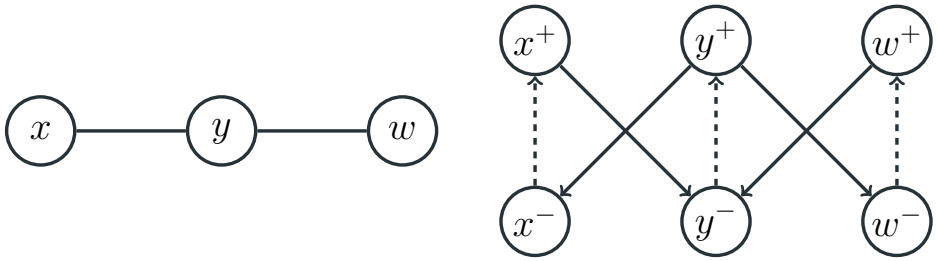
4.4.1 Experimentos com o modelo PLI

Implementamos o modelo PLI apresentado na Seção 4.3 e executamos os experimentos a seguir. Usamos o resolvedor CPLEX da IBM na versão 12.8 com as instâncias do *benchmark*. Configuramos um limite de tempo de 30 minutos e desligamos as funções de *multithread* e *presolve*.

Como o modelo tem um número exponencial de restrições, usamos uma abordagem que relaxa as restrições das Desigualdades (4.3) e as adicionamos depois como *Lazy Constraints*. Esse procedimento insere as restrições relaxadas no modelo matemático apenas quando uma solução inteira é encontrada durante a execução do algoritmo de enumeração, caso alguma delas tenha sido violada pela solução.

Para identificar uma restrição violada, primeiro construímos um grafo direcionado D a partir do grafo de entrada G. Para cada vértice v do grafo G adicionamos dois vérti-

[2]Geramos grafos aleatórios descartando aqueles que não eram conexos.



(a) grafo de entrada (b) grafo direcionado

Figura 4.10: Digrafo usado na separação das soluções inteiras. Nesta figura, exemplificamos a criação do grafo direcionado (Figura 4.10(b)) a partir do grafo não direcionado da entrada (Figura 4.10(a)).

ces v^+ e v^- em D e um arco (v^-, v^+). Para cada aresta $\{u, v\}$ de G adicionamos dois arcos (u^+, v^-) e (v^+, u^-). A Figura 4.10 mostra um exemplo da construção do grafo direcionado.

Seja o grafo da Figura 4.10(a) dado como entrada. Construímos o grafo direcionado da Figura 4.10(b) com o procedimento descrito. Note que, no grafo direcionado, um vértice positivo v^+ só pode ser alcançado a partir do seu vértice negativo correspondente v^-. Além disso, um caminho entre dois vértices que não são vizinhos deve passar por um arco do tipo (v^-, v^+).

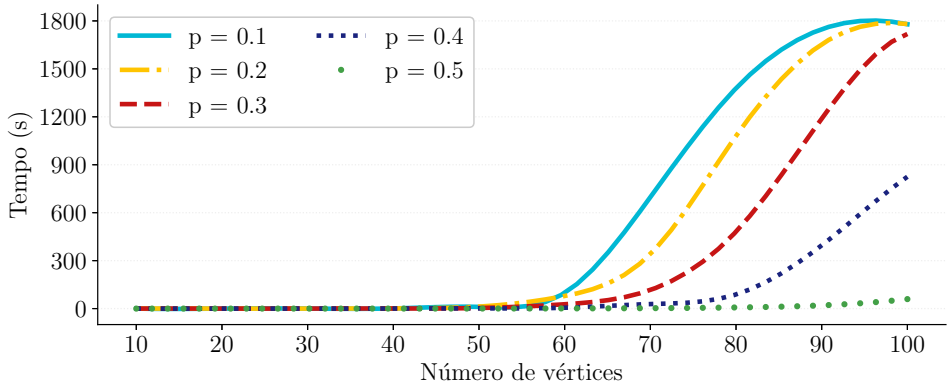
Os próximos passos na identificação de uma restrição violada são repetidos para cada cor usada na coloração inicial do grafo. Seja c uma das cores iniciais do grafo. Definimos um peso para cada arco, de modo que os arcos do tipo (v^-, v^+) recebem o valor de $x_{v,c}$ na solução atual, e os vértices restantes recebem peso 1. Em seguida, executamos um algoritmo de fluxo máximo e corte mínimo usando os pesos definidos como sendo as capacidades dos arcos.

Sejam u e v dois vértices não adjacentes coloridos com a cor c na solução da relaxação. Se o fluxo máximo entre os vértices v^- e u^+ for menor que 1, então os vértices do corte mínimo correspondem a um corte por vértice de uma restrição que está sendo violada. Logo, essa restrição será adicionada ao modelo. Note que os arcos com custo 1 não seriam adicionados no corte mínimo.

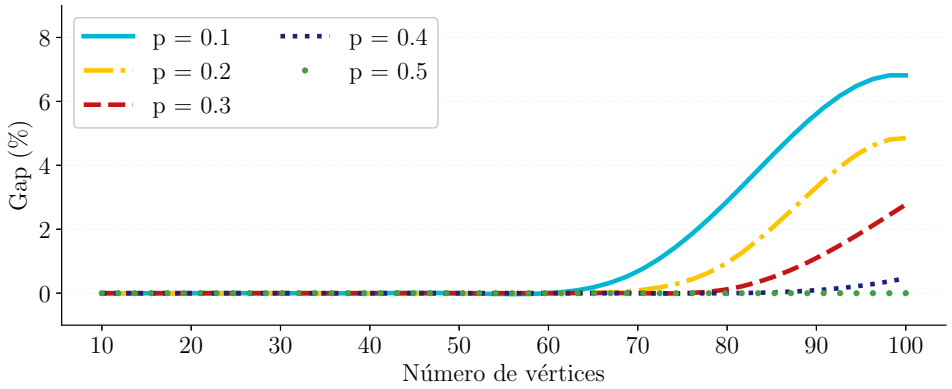
O algoritmo de fluxo máximo e corte mínimo usado nos experimentos foi o algoritmo de Goldberg e Tarjan [20], cuja implementação pode ser encontrada na biblioteca de código aberto LEMON (*Library for Efficient Modeling and Optimization in Networks*) [13].

Na Figura 4.11 resumimos os resultados dos experimentos com o modelo PLI. Em ambos os gráficos, o eixo x indica o número de vértices da instância e cada linha representa um valor diferente para p. Na Figura 4.11(a), temos o gráfico do tempo médio de execução das instâncias. Esse gráfico indica que ao aumentarmos o valor de p, e consequentemente os números de aresta e de cores da coloração inicial, as instâncias são resolvidas mais rapidamente.

Com instâncias que têm mais de 60 vértices, começamos a ter casos em que o resolvedor não termina a execução antes do limite de 30 minutos. Em consequência disso, a instância possui um *gap* positivo. Usamos *gap* para designar a razão entre a melhor solução inteira



(a) tempo



(b) gap

Figura 4.11: Resumo dos experimentos com o modelo de Campêlo *et al.* [7]. Na Figura 4.11(a) apresentamos o tempo médio de execução das instâncias. Já na Figura 4.11(b), apresentamos o *gap* médio, que é positivo quando a instância não termina de executar dentro do tempo limite.

encontrada no momento da interrupção da execução do algoritmo e o ótimo da relaxação do problema.

As médias dos valores de *gap* podem ser vistas da Figura 4.11(b). Note que as instâncias com $p = 0.5$ têm *gap* igual a zero para todos os valores de n, pois, estas terminaram antes do limite. Apesar da média de tempo para $p = 0.4$ ser menor que 30 minutos, ainda tivemos algumas instâncias que possuem *gap* positivo quando $n = 100$, por isso o valor médio de *gap* é diferente de zero.

Na Figura 4.12, apresentamos a média do número de vértices recoloridos pelo modelo PLI. Podemos ver uma relação linear entre o número de vértices e a média de recolorações da instância. Quando $p = 0.1$, a quantidade de vértices recoloridos é maior. Vale notar que, essas também são as instâncias com os maiores valores de *gap*, quando $n > 60$. Apesar disso, o número de vértices que foram recoloridos nessas instâncias seguem a mesma tendência das instâncias com $n \leq 60$, o que indica que as soluções dadas pelo PLI, mesmo tendo um *gap* positivo, estão próximas da média.

Por último, fizemos um experimento para verificarmos se o uso de colorações próprias como coloração inicial tem impacto no tempo de execução. Para isso, criamos outro conjunto de instâncias que têm o mesmo número de cores e a mesma distribuição de vértices por classe de cor. Além disso, todas as classes também são ruins. Para tal,

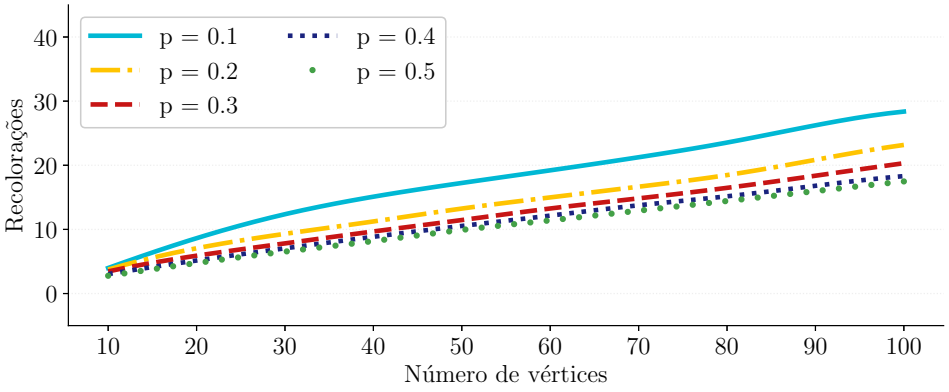


Figura 4.12: Número de vértices recoloridos pelo modelo PLI. Nesse gráfico, cada linha representa um valor distinto de p. Note que, quanto maior o valor de p, maior o número de classes de cor e menor o tamanho da classe.

criamos uma permutação aleatória dos vértices e os distribuímos para cada classe de cor. Caso a coloração tivesse alguma classe convexa, repetimos o processo com uma nova permutação aleatória. Nesse novo conjunto usamos os valores de $n \in \{10, 20, \ldots, 100\}$ e $p \in \{0.1, 0.2\}$. Com valores maiores de p não conseguimos garantir que as classes de cor possuem os mesmos tamanhos e que todas são ruins. Executamos o modelo, e o mesmo conseguiu resolver todas as instâncias antes do tempo limite. Logo, confirmamos que a coloração própria impacta no tempo de resolução das instâncias do problema de recoloração convexa com o modelo PLI.

4.4.2 Experimentos com o GRASP

Nesta seção mostraremos os resultados dos experimentos com o nosso GRASP, considerando as diferentes opções para construção de uma solução, para busca local e para pós-processamento. Os experimentos desta seção usaram o mesmo ambiente de execução e o mesmo conjunto de instâncias que foram testadas nos experimentos com o modelo PLI.

Antes de executarmos qualquer experimento com o GRASP, precisamos primeiro ajustar os dois parâmetros básicos: critério de parada e valor de α. Depois de alguns testes preliminares, decidimos usar o número de iterações igual $2n^2$ para o critério de parada, onde n é o número de vértices da instância. O valor de α indica a porcentagem dos melhores vértices candidatos que devem ser considerados para escolha aleatória. Esse valor foi definido usando o pacote *R Iterated Race Automatic Algorithm Configuration* (`irace`) [27].

Selecionamos 20% das instâncias para usar na execução do pacote `irace`, e as dividimos em um conjunto de treinamento (80%) e teste (20%). Nesses conjuntos, os valores de n e de p estão distribuídos uniformemente. Como temos dois conjuntos de critérios gulosos, o *ratio* e o *union*, executamos o ajuste do parâmetro α para cada um deles. O pacote `irace` retornou os valores 13.95% e 10.23% para o *ratio* e o *union*, respectivamente.

Nos experimentos desta seção, comparamos diferentes abordagens e usamos testes estatísticos que nos dizem se há diferença significativa entre elas. Em especial, aplicamos

Experimento	Teste	Valor	Condição	Rejeita
1. *ratio*: versões puramente gulosa vs aleatorizada	Wilcoxon	$z = -59.85$	$z < -1.96$	Sim
2. *union*: versões puramente gulosa vs aleatorizada	Wilcoxon	$z = -59.06$	$z < -1.96$	Sim
3. *ratio*: vizinhanças para a busca local	Iman-Davenport	$F_F = 174.11$	$F_F > 2.60$	Sim
4. *union*: vizinhanças para a busca local	Iman-Davenport	$F_F = 66.78$	$F_F > 2.60$	Sim
5. *ratio*: *vizinhança estendida* vs *vizinhança simples*	Wilcoxon	$z = -3.16$	$z < -1.96$	Sim
6. *union*: *vizinhança estendida* vs *vizinhança simples*	Wilcoxon	$z = -2.28$	$z < -1.96$	Sim
7. *union*: *none* vs *vizinhança simples*	Wilcoxon	$z = -2.76$	$z < -1.96$	Sim
8. *ratio*: *vizinhança de troca* vs *mutações* e *elite*	Iman-Davenport	$F_F = 124.47$	$F_F > 2.21$	Sim
9. *union*: *vizinhança de troca* vs *mutações* e *elite*	Iman-Davenport	$F_F = 31.38$	$F_F > 2.21$	Sim
10. *ratio*: *mut1* vs *mut2*	Wilcoxon	$z = -0.12$	$z < -1.96$	Não
11. *ratio*: *mut1* vs *mut3*	Wilcoxon	$z = -0.89$	$z < -1.96$	Não
12. *ratio*: *mut2* vs *mut3*	Wilcoxon	$z = -1.01$	$z < -1.96$	Não
13. *union*: *mut1* vs *mut2*	Wilcoxon	$z = -0.17$	$z < -1.96$	Não
14. *union*: *mut1* vs *mut3*	Wilcoxon	$z = -0.02$	$z < -1.96$	Não
15. *union*: *mut1* vs *vizinhança de troca*	Wilcoxon	$z = -2.76$	$z < -1.96$	Sim
16. *union*: *mut2* vs *mut3*	Wilcoxon	$z = -0.19$	$z < -1.96$	Não
17. *union*: *mut2* vs *vizinhança de troca*	Wilcoxon	$z = -2.93$	$z < -1.96$	Sim
18. *union*: *mut3* vs *vizinhança de troca*	Wilcoxon	$z = -2.63$	$z < -1.96$	Sim
19. *union*: *mut1* vs *mut4*	Wilcoxon	$z = -3.05$	$z < -1.96$	Sim
20. *union*: *mut2* vs *mut4*	Wilcoxon	$z = -2.88$	$z < -1.96$	Sim
21. *union*: *mut3* vs *mut4*	Wilcoxon	$z = -2.96$	$z < -1.96$	Sim
22. *ratio* vs *union* vs *composição*	Iman-Davenport	$F_F = 141.88$	$F_F > 3.00$	Sim
23. *composição* vs PLI	Wilcoxon	$z = -12.72$	$z < -1.96$	Sim
24. *composição* vs PLI (mesmos recursos)	Wilcoxon	$z = -22.72$	$z < -1.96$	Sim

Tabela 4.1: Resumo dos testes estatísticos (CR). As linhas desta tabela são referenciadas ao longo da seção para indicar a qual experimento os valores dos testes se referem.

o teste de Wilcoxon quando comparamos duas abordagens e o teste de Iman-Davenport quando comparamos mais de duas abordagens. Esses dois testes possuem a mesma hipótese nula que é a de que não existe uma diferença estatisticamente significativa entre as abordagens.

No teste de Wilcoxon, calculamos o valor da estatística z, com base nos *ranks* das abordagens. Nesse teste, rejeitamos a hipótese nula quando o valor de z for menor que o valor crítico [12]. Em nossos experimentos o número de amostras é 5000, ou seja, o número de instâncias resolvidas por cada abordagem. Considerando esse número de amostras e um nível de confiança de 0.95, rejeitamos a hipótese nula quando $z < -1.95$.

Também usamos os *ranks* das abordagens para calcular o valor da estatística F_F no teste de Iman-Davenport. Nesse caso, o valor crítico depende tanto do número de amostras (N), quanto do número de abordagens comparadas (k). O valor de F_F segue uma distribuição F com $(k-1)$ e $(k-1)(N-1)$ graus de liberdade. Desse modo, podemos computar o valor crítico para cada experimento com essa distribuição, e decidir se podemos ou não rejeitar a hipótese nula [12].

Na Tabela 4.1 resumimos os resultados dos testes estatísticos feitos para os experimentos com o GRASP descritos nessa seção. Ao longo do texto, cada experimento indicará em qual linha da tabela está o seu respectivo resultado do teste estatístico. Na primeira coluna da tabela (**Experimentos**), temos uma breve descrição do experimento. Nas colunas seguintes indicamos: o teste que foi executado (**Teste**); o valor calculado da estatística (**Valor**); a condição para rejeitarmos a hipótese nula (**Condição**); e por último indicamos se a hipótese nula pode ser rejeitada (**Rejeita**), que, em caso afirmativo, indicará que a diferença entre as abordagens é estatisticamente significativa.

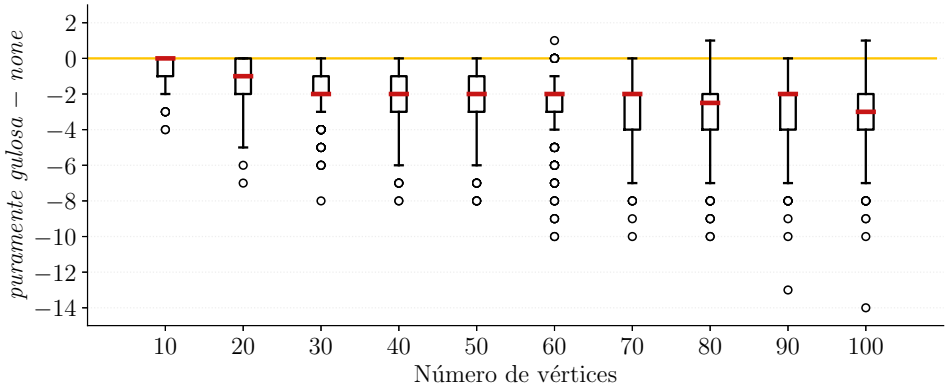
No primeiro experimento com o GRASP, comparamos cada critério (*ratio* e *union*) com sua versão puramente gulosa, isto é, removendo a aleatoriedade e escolhendo sempre o primeiro candidato da lista RCL. Como o foco desse experimento é a aleatoriedade, não usamos a busca local ao final cada iteração do GRASP. Identificamos essa versão com o termo "*none*". Na Figura 4.13, apresentamos os resultados desse experimento.

Nos gráficos da Figura 4.13, o eixo x indica o número de vértices das instâncias e o eixo y indica a diferença no número de vértices recoloridos pelas duas abordagens usando como base a versão com aleatoriedade (*none*). Com isso, um valor negativo nesse eixo indica que a versão *none* recoloriu menos vértices que a versão puramente gulosa. Todos os valores de p estão representados nesse gráfico.

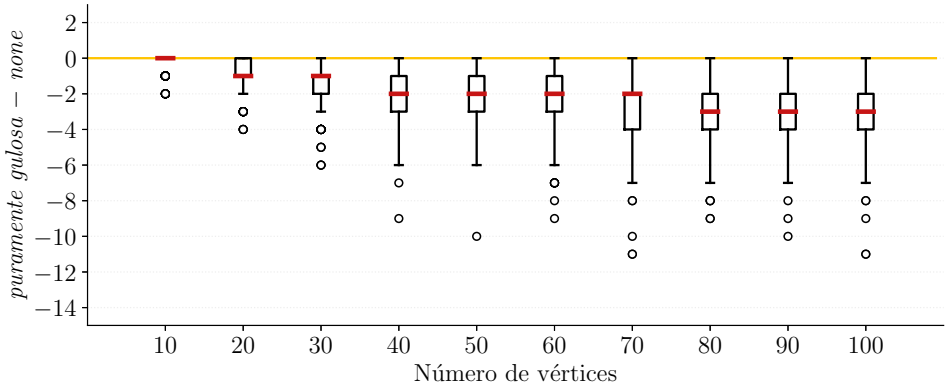
O gráfico da Figura 4.13(a) é referente ao experimento com o conjunto de critérios *ratio*. Nesse experimento, apenas algumas instâncias com 60, 80 e 100 vértices tiveram soluções melhores sem a aleatoriedade. Já o gráfico da Figura 4.13(b) é referente ao experimento com o conjunto de critérios *union*. Nesse experimento, todas as soluções que utilizaram a aleatoriedade do GRASP foram pelo menos tão boas quanto a versão puramente gulosa.

Os resultados dos testes estatísticos para esses dois experimentos estão nas Linhas 1 e 2 da Tabela 4.1. Como a hipótese nula pôde ser rejeitada em ambos os casos, concluímos que a aleatoriedade teve um impacto positivo e significativo na qualidade da solução. Por isso, prosseguimos os experimentos usando essa abordagem.

O próximo experimento compara as diferentes vizinhanças para a busca local. Para



(a) ***ratio***



(b) ***union***

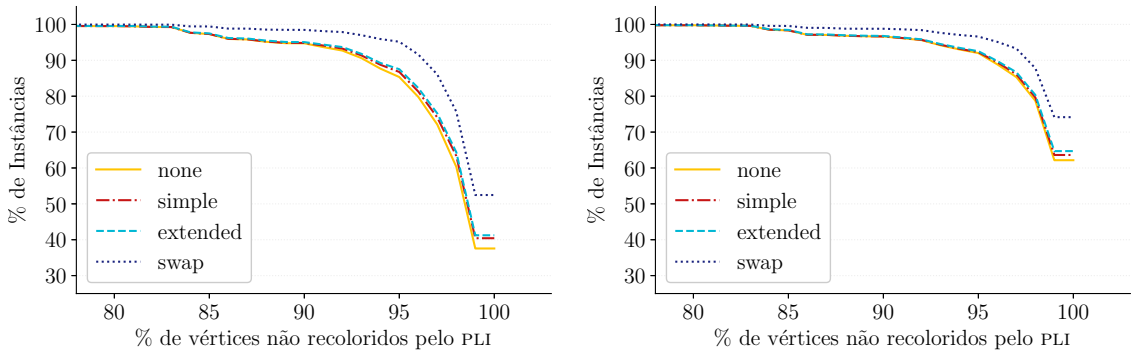
Figura 4.13: Comparando as versões puramente gulosas com as versões que usam aleatoriedade. Nos gráficos dessa figura, um valor negativo no eixo y indica que a versão que usa aleatoriedade (*none*) recoloriu menos vértices.

cada conjunto de critérios, executamos o GRASP com a busca local ao final de cada iteração. Usamos o termo *none* para nos referirmos à versão do GRASP sem busca local e os termos *simple*, *extended* e *swap* para nos referirmos às versões com busca local usando a *vizinhança simples*, a *vizinhança estendida* e a *vizinhança de troca*, respectivamente.

Comparamos cada versão da busca local com a melhor solução dada pelo modelo PLI e criamos os gráficos da Figura 4.14. Se em uma certa instância, o GRASP manteve a cor de 8 vértices e o PLI manteve a cor de 10, dizemos que a proporção dessa instância será de $\frac{8}{10} = 0.8$, isto é, o custo da solução do GRASP conseguiu alcançar 80% daquele da solução do PLI. Assim os eixos y desses gráficos indicam a porcentagem de instâncias que atingiram pelo menos a proporção x da solução dada pelo PLI, para os conjuntos *ratio*, da Figura 4.14(a), e *union*, da Figura 4.14(b).

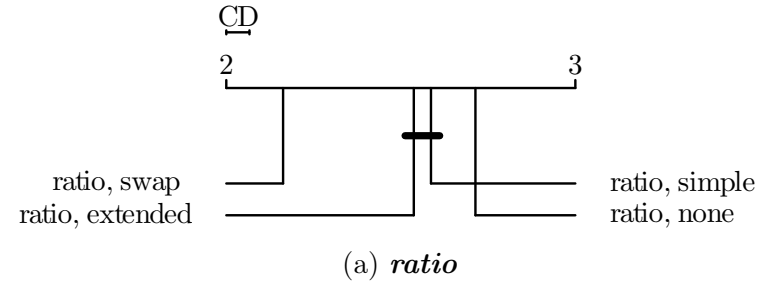
Em ambos os experimentos todas as instâncias mantiveram a cor de pelo menos 80% do número de vértices mantidos pelo PLI. As vizinhanças têm comportamentos parecidos em ambos os conjuntos, sendo a busca local com a *vizinhança de troca* a abordagem com os melhores resultados, pois obteve soluções mais próximas daquelas dadas pelo PLI. Outra informação importante que podemos extrair dos gráficos, é que o conjunto *union* obteve mais soluções próximas daquelas do PLI.

Nas Linhas 3 e 4 da Tabela 4.1, temos os resultados dos testes estatísticos para os

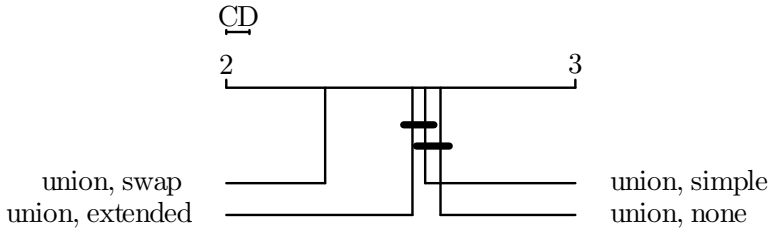


(a) ***ratio*** (b) ***union***

Figura 4.14: Comparando a busca local usando diferentes vizinhanças com o PLI. Nos gráficos, cada linha corresponde a uma abordagem do GRASP.



(a) ***ratio***



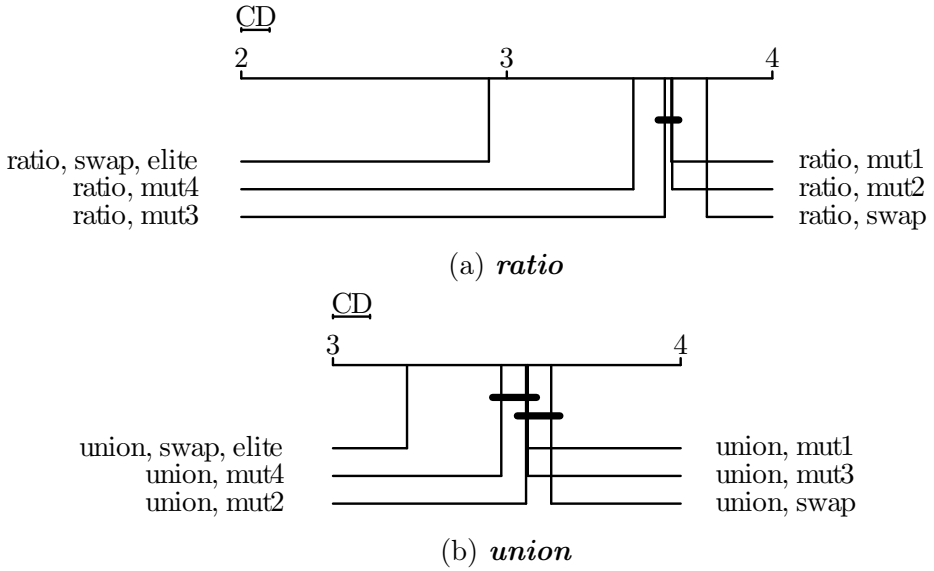
(b) ***union***

Figura 4.15: Diferença crítica entre as abordagens com busca local. Nesses gráficos, caso não exista uma diferença estatísticamente significativa entre duas abordagens, elas estarão conectadas por uma linha horizontal.

experimentos com as buscas locais. Como a hipótese nula pôde ser rejeitada em ambos os casos, executamos um segundo teste estatístico, chamado de teste de Nemenyi [12], com esses dados para verificarmos entre quais versões a diferença é realmente significativa. Vale notar que o teste de Iman-Davenport verifica se existem diferenças significativas no conjunto inteiro, por isso usamos também o segundo teste.

O teste de Nemenyi possui uma representação gráfica (Figura 4.15). Nessa representação, as abordagens são representadas por uma linha vertical, e, caso as diferenças entre elas não sejam significativas, haverá uma linha horizontal conectando-as. Com esse teste, identificamos quais abordagens não são equivalentes, contudo, ainda é necessário um teste mais robusto para confirmar que as abordagens marcadas são de fato equivalentes [12].

Nos gráficos das Figuras 4.15(a) e 4.15(b) temos os resultados do teste de Nemenyi para os conjuntos *ratio* e *union*, respectivamente. Esses gráficos indicam que as abordagens com busca local usando a *vizinhança simples* e a *vizinhança estendida* são equivalentes em



(a) ***ratio***

(b) ***union***

Figura 4.16: Diferença crítica entre as abordanges com pós-processamentos. Cada gráfico é referente um dos conjuntos de critérios (*ratio* e *union*). As abordagens que não estão conectadas por uma linha horizontal possuem uma diferença estatisticamente significativa.

ambos os conjuntos. As abordagens sem busca local e com busca local usando a *vizinhança simples* também são marcadas como equivalentes, com o conjunto *union*. Contudo, ao analisarmos esses pares individualmente com o teste de Wilcoxon, podemos rejeitar a hipótese nula, como mostram as Linhas 5, 6 e 7 da Tabela 4.1.

As abordagens que aparacem nos gráficos das Figuras 4.15(a) e 4.15(b) são ordenadas de modo que a aquela com o maior *rank* médio está mais à esquerda. Esses resultados são condizentes com o que é apresentado nos gráficos das Figuras 4.14(a) e 4.14(b), que indicam que as abordagens que usam a busca local com a *vizinhança de troca* obtiveram os melhores resultados.

Os experimentos seguintes investigam o impacto do uso dos procedimentos de pós-processamento. Executamos os experimentos apenas com as versões que usam a *vizinhança de troca*, pois obtiveram os melhores resultados. O primeiro pós-processamento usado é a mutação, que é aplicado ao final de cada iteração do GRASP. Definimos quatro versões da mutação, chamadas de *mut1*, *mut2*, *mut3* e *mut4*. A base da mutação consiste em remover a cor dos vértices de uma determinada classe, com exceção de um. Com isso, usamos os seguintes critérios para a escolha das classes:

- *mut1*: a maior classe;
- *mut2*: a menor classe;
- *mut3*: uma classe aleatória;
- *mut4*: uma classe por vez, recuperando a recoloração inicial após a execução da busca local.

Nesse experimento, também usamos o pós-processamento *elite*. Definimos $m = 5$ para o tamanho do conjunto de soluções da elite, logo, o modelo PLI é executado no

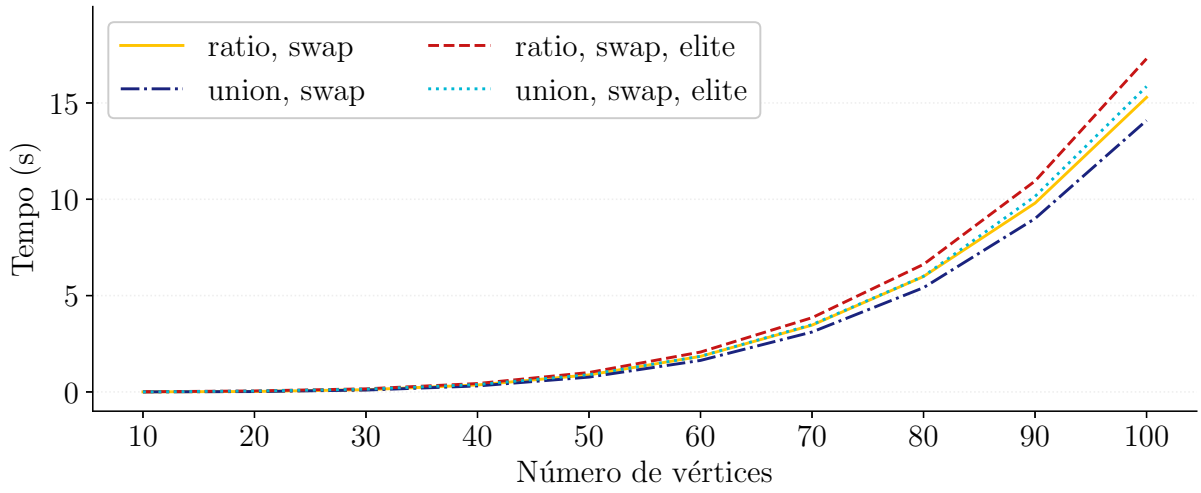


Figura 4.17: Tempo médio de execução com o pós-processamento *elite*. Cada linha do gráfico se refere a uma versão do GRASP.

máximo $\binom{5}{2} = 10$ vezes após todas as $2n^2$ iterações do GRASP terminarem. Limitamos o tempo de cada execução do PLI a 10 segundos.

Executamos os experimentos com os pós-processamentos para ambos os conjuntos, *ratio* e *union*. Como existem diferenças significativas entre as abordagens usando os pós-processamentos (*mutações* e *elite*) e a busca local com a *vizinhança de troca* (Linhas 8 e 9 da Tabela 4.1), criamos os gráficos da Figura 4.16. Neles, vemos que o pós-processamentos *elite* obteve os melhores *ranks* médios.

Pelo gráfico da Figura 4.16(a) temos que as 3 abordagens *mut1*, *mut2* e *mut3* são equivalentes, quando combinadas com o *ratio*. Executando os testes dois a dois (das Linhas 10, 11 e 12 da Tabela 4.1), ainda não podemos afirmar que a diferença entre essas abordagens é significativa.

O gráfico da Figura 4.16(a) é referente ao conjunto *union*. Nele temos que as *mutações* $1, 2,$ e 3 são equivalentes a não usar a mutação, e também são equivalentes a *mut4*. Entretanto, podemos confirmar apenas que as abordagens *mut1*, *mut2* e *mut3* são equivalentes com ambos os conjuntos de critérios (Linhas 13 – 21 da Tabela 4.1).

Como as abordagens que usam a *elite* obtiveram os melhores resultados, investigamos o impacto do uso desse pós-processamento no tempo de execução das instâncias. Apresentamos a média do tempo de execução das instâncias da Figura 4.17. No gráfico dessa figura, o eixo y representa a média de tempo em segundos de todas as instâncias com x vértices.

Cada linha do gráfico da Figura 4.17 representa uma versão do GRASP. O aumento no tempo de execução foi bem pequeno, em média não foi maior que dois segundos. Verificamos que os tamanhos das interseções de duas soluções *elite* são em média 70% do número de vértices das instâncias, e esse valor teve um pequeno crescimento com instâncias maiores. Isso significa que o modelo PLI é executado com grande parte das variáveis já fixadas, o que reduz bastante o tamanho da entrada. Também verificamos que todas as execuções do modelo nessa fase do GRASP terminaram antes do tempo limite de 10 segundos.

A partir deste ponto no texto, quando usamos os termos *ratio* e *union*, estamos nos

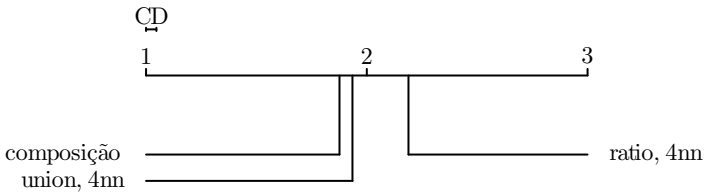


Figura 4.18: Diferença crítica entre *ratio*, *union* e *composição*. Como não existem linhas horizontais conectando abordagens, temos que não existem equivalências no conjunto analisado.

referindo a versão que apresentou os melhores resultados nos experimentos anteriores, ou seja, as versões que usam a aleatorização do GRASP, a busca local com a *vizinhança de troca* e o pós-processamento *elite*. O próximo experimento foca na comparação das duas abordagens *ratio* e *union*.

Uma forma simples de comparar o *ratio* e o *union* é contar quantas vezes um foi melhor que o outro. Ao fazermos isso, temos que os dois conjuntos obtiveram soluções com o mesmo custo em 75.78% das instâncias, o *ratio* obteve a melhor resposta em 4.24% das instâncias e o *union* em 19.98% das instâncias.

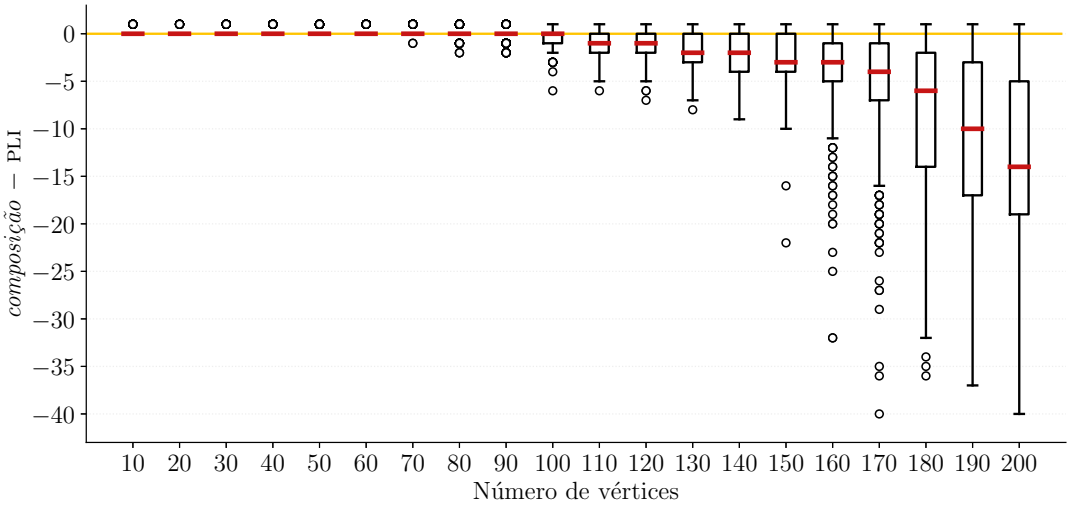
Com os gráficos da Figura 4.14, que tratam dos experimentos com as vizinhanças da busca local, já tínhamos um indicativo de que o *union* tem um desempenho melhor que o *ratio*. Contudo, ainda temos alguns casos em que o *ratio* foi estritamente melhor que o *union*, além do grande número de empates. Assim, investigamos se usar os dois conjuntos traz uma melhora significativa para o algoritmo, isto é, se a quantidade de instâncias em que o *ratio* obteve o melhor resultado é significativa.

Chamamos de *composição* a abordagem que combina duas execuções consecutivas do algoritmo GRASP, uma vez com o *ratio* e outra com o *union*. Desse modo, o tempo de execução dessa abordagem é a soma das duas execuções e a solução retornada é a melhor das duas execuções. Como a *composição* executa $4n^2$ iterações do GRASP, não a comparamos com as soluções encontradas com os experimentos das abordagens de pós-processamento, e sim com uma nova execução do *ratio* e do *union* que também usa $4n^2$ iterações, e identificamos essa versão com o sufixo $4nn$.

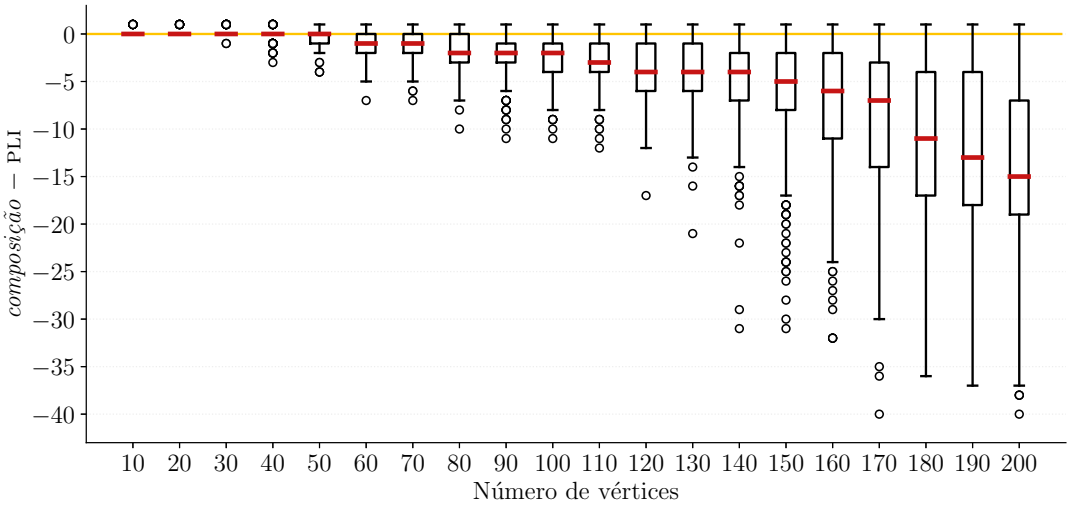
A diferença entra essas três abordagens é estatisticamente significativa (Linha 22 da Tabela 4.1). Criamos o gráfico da Figura 4.18, e atestamos que a diferença entre quaisquer duas abordagens também é estatisticamente significativa. Além disso, a composição possui o melhor *rank* médio, e a consideramos a melhor versão do GRASP a partir desse ponto.

Para os experimentos seguintes, adicionamos 100 instâncias para cada par (n, p), com $n \in \{110, 120, \ldots, 200\}$ e $p \in \{0.1, 0.2, \ldots, 0.5\}$. Executamos o modelo PLI para obter um limitante dessas instâncias. Nessa execução do modelo, também usamos um limite de 30 minutos. Comparamos as soluções dadas pelo GRASP (versão *composição*) com as soluções dadas pelo modelo PLI. O resolvedor não retornou uma solução inteira para sete das novas instâncias, e por isso, nesses casos, usamos apenas o limitante da relaxação linear para a comparação das soluções do GRASP.

Na Figura 4.19 temos dois gráficos que comparam as soluções do GRASP e do PLI. Nesses gráficos, o eixo x indica o número de vértices da instância, e o eixo y indica a diferença no número de recolorações, onde um valor negativo indica o número de vértices



(a) PLI executou por 30 minutos



(b) PLI executou com o mesmo tempo do GRASP.

Figura 4.19: Comparação do número de recolorações do GRASP com o PLI. Nesses gráficos, um valor negativo no eixo x indica que o GRASP recoloriu menos vértices que o PLI.

que o GRASP recoloriu a menos que o modelo PLI.

No gráfico da Figura 4.19(a), o limite de tempo do PLI foi de 30 minutos. Note que com $n > 60$, temos casos em que o GRASP conseguiu uma solução melhor que a do PLI. Isso é possível, pois nessas instâncias o PLI teve sua execução interrompida pelo limite de tempo. Para todos os tamanhos de instâncias, a heurística recoloriu no máximo um vértice a mais que o modelo. Essas instâncias correspondem a 8% do total. Além disso, em 39% das instâncias, o GRASP recoloriu menos vértices que o limitante do modelo. Para o restante das instâncias as duas abordagens recoloriram o mesmo número de vértices.

Para obter o gráfico da Figura 4.19(b), executamos um novo experimento que limita o tempo de execução do PLI ao tempo de execução do GRASP. Fazemos isso para comparar o desempenho das duas abordagens, quando dados os mesmos recursos. O gráfico é bem parecido com o anterior, com a diferença de que o GRASP aumentou a diferença no número de vértices recoloridos para as instâncias com n maior ou igual a 60.

Por último executamos testes estatísticos comparando o GRASP com o PLI. No primeiro teste usamos a melhor solução do PLI, limitado a 30 minutos (Linha 23 da Tabela 4.1) e no segundo usamos a solução PLI quando este dispõe dos mesmos recursos computacionais que o GRASP (Linha 24 da Tabela 4.1). Em ambos os casos a hipótese nula pôde ser rejeitada e o GRASP obteve o melhor *rank* médio. Assim, temos que GRASP obteve a melhor solução no maior número de instâncias e possui uma diferença estatisticamente significativa em relação ao modelo PLI.

4.5 Conclusões

Neste capítulo, focamos no problema de Recoloração Convexa (CR) em grafos gerais. Propusemos uma heurística baseada no *Greedy Randomized Adaptive Search Procedure* (GRASP) para essa versão do problema. Implementamos um modelo de Programação Linear Inteira (PLI) que foi proposto por Campêlo *et al.* [5]. Realizamos experimentos com as duas abordagens e usamos o modelo PLI para verificar a qualidade das soluções dadas pelo GRASP.

No melhor do nosso conhecimento, não existem experimentos computacionais do problema de recoloração convexa com grafos gerais reportados na literatura. Por isso, criamos um *benchmark* de instâncias que foram usados nos experimentos com o modelo PLI e com a heurística GRASP. O modelo PLI de Campêlo *et al.* [5] possui um número exponencial de restrições, por isso, implementamos um procedimento de separação das soluções inteiras.

Desenvolvemos dois conjuntos de critérios para criar uma solução gulosa com o GRASP (*ratio* e *union*), três vizinhanças para a busca local (*vizinhança simples*, *vizinhança estendida* e *vizinhança de troca*) e dois procedimentos de pós-processamento (*mutação* e *elite*). Executamos experimentos com as versões do GRASP e, com a ajuda de testes estatísticos, determinamos que a melhor versão combina os dois conjuntos de critérios gulosos *ratio* e *union*, usa a *vizinhança de troca* na busca local (que é executada em cada iteração do GRASP) e o pós-processamento com as soluções *elite*, que é executado após a última iteração do GRASP.

Ao compararmos as soluções dadas pelo GRASP com as melhores soluções dadas pelo

PLI para as 10000 instâncias do *benchmark*, verificamos que a melhor versão do GRASP retornou soluções boas para as instâncias propostas. Tendo em vista que as soluções do GRASP recoloriram no máximo 1 vértice a mais que o PLI, e isso aconteceu em apenas 8% das instâncias. Nas instâncias restantes, o GRASP recoloriu pelo menos o mesmo número de vértices que o PLI, sendo que a solução do GRASP foi estritamente melhor em 39% das instâncias.

O GRASP apresentado neste capítulo pode ser usado para outras versões do problema de Recoloração Convexa com pouca ou nenhuma alteração nos processos de criação de solução, busca local e pós-processamento. Além disso, o processo ainda pode ser melhorado combinando-o com outras meta-heurísticas, como a *Variable Neighborhood Search* (VNS), que pode ser incorporada ao GRASP usando as vizinhanças já definidas.

Os grafos usados no *benchmark* são grafos aleatórios criados usando o procedimento de Erdös-Rényi. Esses grafos possuem características bem comportadas, como o número cromático. Outros experimentos com o GRASP podem ser feitos alterando a classe do grafo para *grids*, por exemplo.

Capítulo 5

Recoloração Convexa Restrita

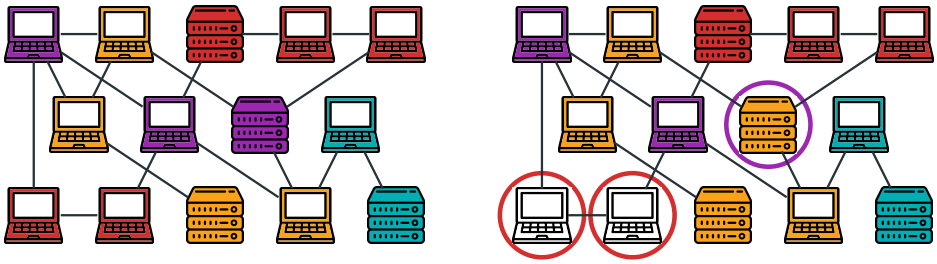
Neste capítulo abordamos o problema de Recoloração Convexa Restrita. Na Seção 5.1 definimos essa versão do problema de recoloração. Na Seção 5.2 listamos as modificações necessárias para adaptar as abordagens do CR para a versão restrita. Na Seção 5.3 descrevemos os experimentos com as abordagens adaptadas e discutimos seus resultados. Por último, na Seção 5.4, apresentamos as conclusões e apontamos possíveis trabalhos futuros.

5.1 Introdução

Kammer e Tholey [22] propuseram duas versões do problema de recoloração convexa com aplicações em redes de computadores. A primeira versão é chamada de Recoloração Convexa Restrita, ou *Minimum Restricted Recoloring Problem* (MRRP), que define um conjunto de vértices que não podem ser recoloridos, isto é, podem apenas ser descoloridos. Já a segunda versão é chamada de Recoloração Convexa em Blocos, ou *Minimum Block Recoloring Problem* (MBRP), e altera a função de custo do problema.

Na Recoloração Convexa Restrita, usamos as cores para representar serviços e dividimos o conjunto de vértices entre clientes e servidores. Os vértices clientes estão interessados em usar um serviço, determinado pela sua cor inicial. Já os vértices servidores podem prover um serviço qualquer para os clientes. O objetivo desse problema é que todos os vértices relacionados ao mesmo serviço estejam conectados. Além disso, essa conexão não deve depender de vértices relacionados a outros serviços. Isso implica que os vértices que usam o mesmo serviço induzem um subgrafo conexo.

Por causa da natureza do problema, um vértice cliente não pode receber um serviço diferente daquele que deseja. De modo que, há apenas duas possibilidades para esses vértices: ser atendido, que implica que o vértice mantém sua cor original, ou não ser



(a) (b)

Figura 5.1: Exemplo de instância para o MRRP. Na primeira figura, temos um exemplo de entrada, onde apenas uma classe de cor é convexa. Na figura seguinte, temos uma recoloração convexa que respeita a restrição dos vértices clientes para o problema.

atendido, que implica na sua descoloração. Essa é a principal característica do MRRP.

Definição 5.1 RECOLORAÇÃO CONVEXA RESTRITA – MRRP

Entrada: um grafo $G = (V_r \cup V_c, E)$, onde V_r é o conjunto de vértices servidores e V_c é conjunto dos vértices clientes, um conjunto de cores $\mathcal{C}$, uma coloração parcial C e uma função de custo w.

Objetivo: Encontrar uma coloração boa C' com custo mínimo, $\min(\sum_{v \in R_C(C')} w(v))$, tal que para todo vértice cliente $v \in V_c$, $C'(v) = C(v)$ ou $C'(v) = \emptyset$.

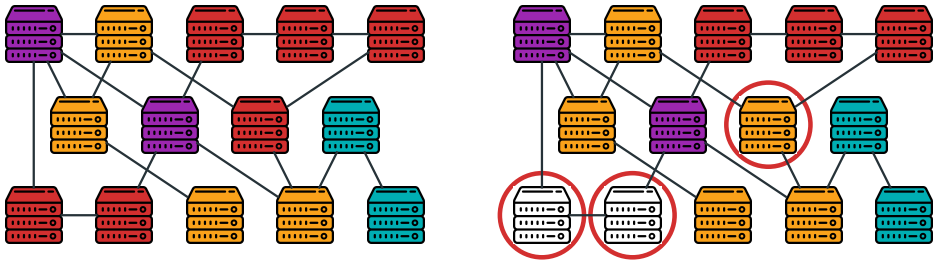
Saída: Uma coloração boa C'.

A Definição 5.1 apresenta o MRRP formalmente. Note que apenas a saída é igual às outras versões dos problemas de recoloração convexa estudadas neste trabalho. A entrada do problema agora deve indicar quais os papéis dos vértices e o objetivo deve respeitar a restrição imposta nos vértices clientes.

Usamos a Figura 5.1 para dar um exemplo de instância para o MRRP. Suponha que o grafo colorido da Figura 5.1(a) é dado como entrada. Nesse grafo temos quatro servidores, um em cada classe de cor (●, ●, ● e ●). Na Figura 5.1(b) apresentamos uma possível recoloração convexa restrita para a instância da figura anterior. Nessa recoloração, a cor de três vértices foram alteradas: dois clientes vermelhos (●) foram descoloridos e um servidor roxo (●) foi recolorido.

Note que, sem considerar os papéis dos vértices, seria possível encontra uma recoloração com custo dois. Essa recoloração trocaria a cor do servidor roxo para amarelo e a cor do cliente roxo da segunda linha para vermelho.

A segunda versão do problema proposta por Kammer e Tholey [22] é a Recoloração Convexa em Blocos, que é apresentada na Definição 5.2. Nessa versão, não há distinção entre os vértices, sendo todos considerados como servidores. O custo de recoloração não é dado pelo número de vértices que foram recoloridos, e sim pelo número de classes de



(a) (b)

Figura 5.2: Exemplo de instância para o MBRP. Na primeira figura, temos um exemplo de entrada. Na figura seguinte, temos uma recoloração convexa que troca a cor de três vértices, mas tem custo um.

cores que perderam vértices na recoloração.

Definição 5.2 RECOLORAÇÃO CONVEXA EM BLOCOS – MRRP

Entrada: um grafo qualquer G, um conjunto de cores $\mathcal{C}$, uma coloração parcial C e uma função de custo w.

Objetivo: encontrar uma coloração boa C' com custo mínimo, $\min(\sum_{c \in B(C')} w'(c))$, com $B(C')$ sendo o conjunto de cores que perderam pelo menos um vértice em C', $B(C') = \{c \in C | \ \exists\ v \in V$ tal que $C(v) = c$ e $C'(v) = c'\}$, com $c \neq c'\}$

Saída: coloração boa C'.

Temos um exemplo de instância do MBRP na Figura 5.2. Seja o grafo colorido da Figura 5.2(a) o grafo da entrada. Recolorimos três vértices servidores nessa instância, como mostra a Figura 5.2(b). Contudo, o custo da recoloração é apenas um, já que a classe vermelha (●) foi a única que perdeu vértices na recoloração. Se considerássemos a versão do problema de recoloração convexa com o custo uniforme, poderíamos encontrar uma recoloração que troca a cor de apenas dois vértices para a instância da Figura 5.2(a).

Além de introduzir duas novas versões do problema de recoloração convexa, Kammer e Tholey [22] também apresentaram resultados sobre a complexidade desses problemas em grafos com largura arbórea limitada. Foi provado que o MRRP é NP-Difícil quando o número de vértices por classe de cor é maior que 3. Para o MBRP, foi apresentado um algoritmo polinomial.

Neste trabalho, adaptamos o modelo de Campêlo *et al.* [6] e a nossa heurística GRASP para tratar do MRRP. A seguir, detalhamos as modificações nas abordagens e apresentamos os experimentos que efetuamos com as abordagens adaptadas. Como não existem resultados computacionais reportados para o problema, também definimos conjuntos de instâncias de teste para os algoritmos.

5.2 Adaptação para o MRRP

Nesta seção, apresentamos as mudanças feitas para adaptar o modelo PLI proposto por Campêlo *et al.* [6] e a nossa heurística GRASP, de modo que estas abordagens produzam soluções válidas para o MRRP.

Para adaptar o modelo PLI, adicionamos as restrições descritas nas Equações (5.1). Seja o conjunto de vértices clientes denominado por V_c, a cor inicial de v é dada por $C(v)$. No modelo de Campêlo *et al.* [6], a variável $x_{v,c}$ recebe 1 se o vértice v é colorido com a cor c, e 0 caso contrário. Então, acrescentamos ao MRRP as seguintes restrições:

$$x_{v,c} = 0 \quad \forall v \in V_c, \forall c \in \mathcal{C} \backslash \{C(v)\}. \tag{5.1}$$

As Equações (5.1) restringem que um vértice cliente possa receber uma cor diferente daquela com a qual foi inicialmente colorido. Note que se o vértice for descolorido, então $x_{v,c}$ é igual a 0 para todo c. A seguir definimos as mudanças no GRASP.

No algoritmo que constrói uma solução (Algoritmo 2), alteramos a operação de contração para tratar dos papéis dos vértices. Definimos que, se uma aresta conecta dois vértices com papéis diferentes, o vértice que permanece no grafo receberá o papel de cliente.

No critério *ratio* definimos que o peso de um vértice cliente que é candidato a recoloração igual a n, isto é, colocamos um peso suficientemente grande para que na ordenação esses vértices fiquem no final da lista RCL. Isso faz com que os vértices servidores sejam escolhidos primeiro. Além disso o *fator de proporção* de um vértice cliente é 0, pois ele não poderá ser usado para conectar componentes monocromáticas.

Já no critério *union*, o custo de recolorir um vértice cliente será sempre 1 e o *fator de união* será sempre 0, já que, se for recolorido, o vértice cliente receberá a cor $\emptyset$. Com esta última mudança, também reduzimos as chances de um vértice cliente ser escolhido primeiro no critério *union*. Em ambos os critérios, verificamos se o vértice sorteado para recoloração é um cliente antes de selecionar uma nova cor.

Dentre as vizinhanças para a busca local, apenas a *vizinhança simples* não precisa ser alterada. Na *vizinhança estendida* restringimos que o caminho que conecta um vértice descolorido com um vértice colorido com sua cor original contenha apenas vértices servidores.

Considerando a *vizinhança de troca*, fizemos a alteração descrita a seguir. Seja v um vértice do conjunto S, isto é, v foi recolorido em C' e v não era um ponto de articulação na sua classe em C'. Na *vizinhança de troca* para o CR, procuramos um caminho (v, w, u) para cada v, tal que u está colorido com a cor original de v ($C'(u) = C(v)$) e w não é um ponto de articulação. Encontrado esse caminho, recolorimos v e w com a cor de u. Contudo, para gerar uma solução do MRRP, adicionamos a restrição de que o vértice w também não pode ser um cliente.

Como acontece com a *vizinhança simples*, os procedimentos de pós-processamento não precisam ser alterados.

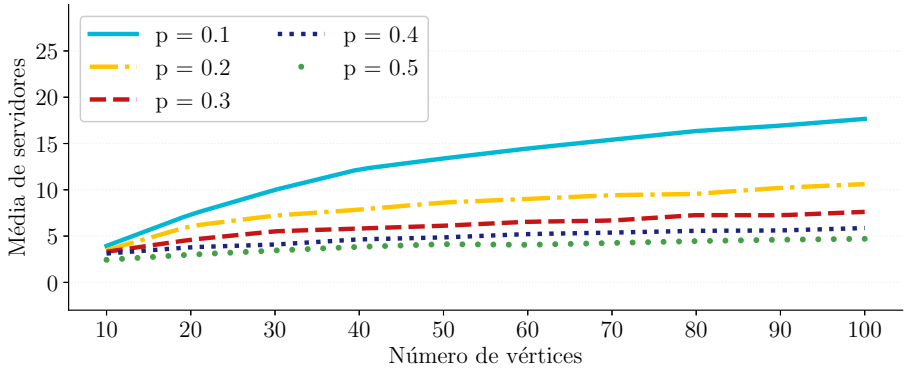


Figura 5.3: Tamanho médio do conjunto dominante (servidores).

5.3 Experimentos computacionais

Os experimentos reportados nessa seção também foram executados numa máquina com processador Intel^{R} Core^{TM} i7 3.40GHz, 32GB de memória e sistema operacional Ubuntu 18.04 LTS. As modificações em ambas as abordagens foram implementadas no código já existente. Usamos o resolvedor de programação linear inteira `IBM CPLEX` na versão 12.8, com as funções de *presolve* e de paralelismo do `CPLEX` desativadas.

As instâncias foram criadas a partir das instâncias propostas para o CR, selecionando um subconjunto dos vértices para receber o papel de servidor. Usamos apenas as instâncias de tamanho 10 a 100. Optamos por atribuir os vértices de um conjunto dominante aos vértices servidores para evitar casos em que clientes estão conectados apenas a outros clientes, garantindo, assim, que todos os vértices tenham a possibilidade de manter sua cor inicial. A seguir, descrevemos o procedimento usado para criar o conjunto dominante.

Seja $\mathcal{Y}$ o conjunto dominante, inicialmente vazio. Selecionamos o vértice v de maior grau e o adicionamos a $\mathcal{Y}$. Em seguida removemos v e todos os seus vizinhos do grafo. Repetimos o processo até que não existam mais vértices no grafo.

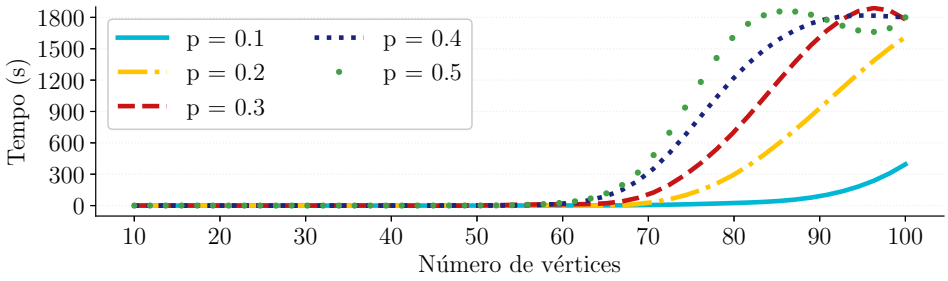
Na Figura 5.3, apresentamos a média do tamanho do conjunto dominante para cada valor de n e p, onde n é o número de vértices da instância e p é a probabilidade de uma aresta ser adicionada ao grafo usando o método de Erdös-Rényi. Vale lembrar que para cada par (n, p) temos 100 instâncias. Nesse gráfico, ao tornarmos o grafo mais denso, temos uma redução no tamanho médio do conjunto dominante, de modo que este acaba ficando bem pequeno quando $p = 0.5$.

5.3.1 Experimentos com o PLI modificado

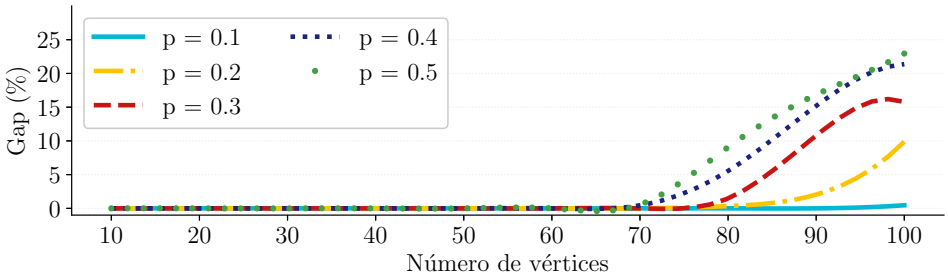
O primeiro experimento com o MRRP foi executar o modelo PLI com as novas restrições. A Figura 5.4 apresenta os resultados desse experimento, cujo tempo limite de execução foi de 30 minutos.

No gráfico da Figura 5.4(a) apresentamos o tempo médio de execução das instâncias agrupados por valor de p. Ao contrário do que aconteceu com o CR, esse gráfico mostra que, ao aumentarmos o valor de p, as instâncias usam em média mais tempo de execução. Note que essas também são as instâncias que possuem os menores números de servidores.

Como houve instâncias que não terminaram a execução dentro desse tempo limite,



(a) tempo



(b) gap

Figura 5.4: Resumo dos experimentos com modelo PLI modificado para o MRRP. No gráfico da Figura 5.4(a), temos o tempo médio de execução das instâncias. No gráfico da Figura 5.4(b), temos o *gap* médio das instâncias.

algumas delas possuem um *gap* positivo, como mostra o gráfico da Figura 5.4(b). Vale notar que o gap médio das instâncias, quando comparado com o CR, também é maior.

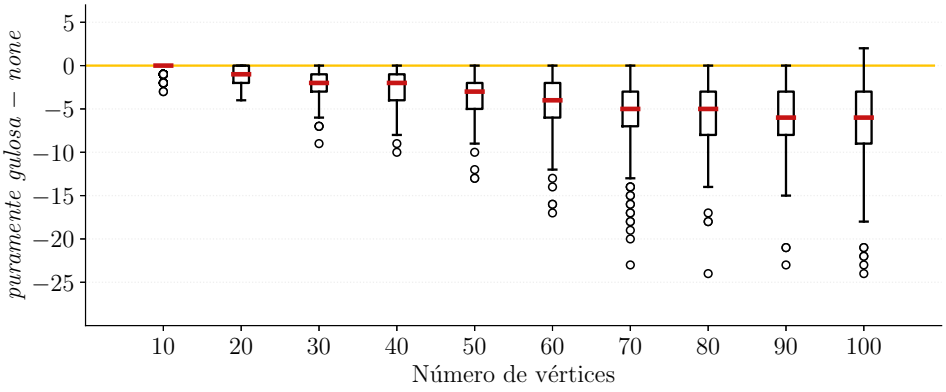
5.3.2 Experimentos com o GRASP modificado

Com as alterações, descritas na seção anterior, para as versões do GRASP implementadas, usamos novamente o pacote `irace` para configurar o parâmetro α. Para o conjunto *ratio* definimos $\alpha = 23.29\%$ e para o conjunto *union* definimos $\alpha = 32.09\%$. Além disso, mantivemos o número de iterações igual a $2n^2$, onde n é o número de vértices da instância.

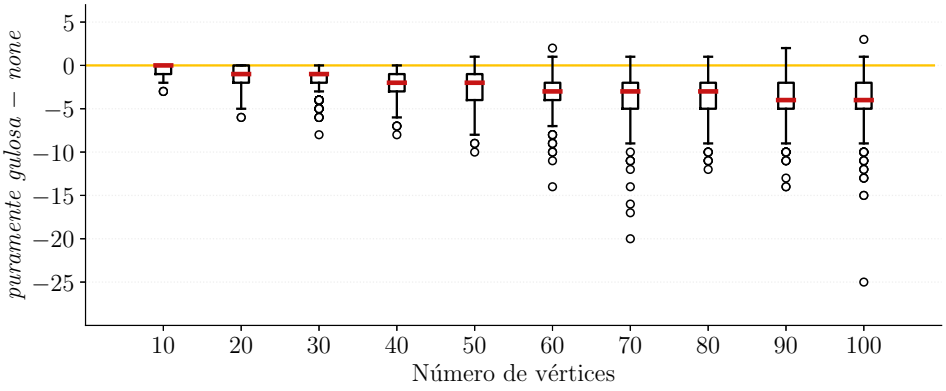
Resumimos os resultados dos testes estatísticos dos experimentos desta seção na Tabela 5.1, que pode ser encontrada no final desta seção. As linhas dessa tabela serão referenciadas ao longo do texto, como no capítulo anterior.

No primeiro experimento com o GRASP para o MRRP, verificamos o impacto do uso da aleatoriedade com a abordagem puramente gulosa. Nesse experimento não utilizamos a busca local ao final de cada iteração do GRASP, e chamamos essa versão de *none*. Nos gráficos da Figura 5.5, um valor negativo no eixo y indica que a versão que usa aleatoriedade recoloriu menos vértices que a versão puramente gulosa.

Os gráficos das Figuras 5.5(a) e 5.5(b), são referentes às versões que usam o conjunto de critérios gulosos *ratio* e *union*, respectivamente. Com esses gráficos, temos que a maior parte das instâncias tiveram soluções melhores com a aleatoriedade. Em contraste com a versão do CR, o conjunto *union* teve mais instâncias nas quais a versão gulosa obteve as melhores soluções. Os resultados dos testes estatísticos com essas versões nos mostram que há uma diferença estatisticamente significativa entre as abordagens (Linhas 1 e 2 da Tabela 5.1).



(a) ***ratio***



(b) ***union***

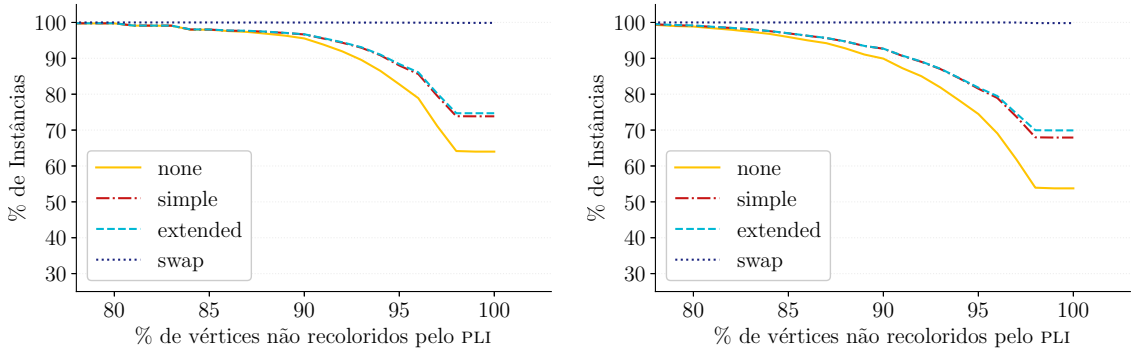
Figura 5.5: Comparação das versões puramente gulosas e aleatorizadas do GRASP para o MRRP. Nos gráficos das Figuras 5.5(b) e 5.5(b), um valor negativo indica que a versão aleatorizada recoloriu menos vértices que a versão puramente gulosa.

No próximo experimento, comparamos as versões das vizinhanças para a busca local com a versão que não usa busca local. Usamos o termo *none* para nos referirmos à versão do GRASP sem busca local e os termos *simple*, *extended* e *swap* para nos referirmos às versões com busca local usando a *vizinhança simples*, a *vizinhança estendida* e a *vizinhança de troca*, respectivamente.

Para comparar a qualidade das soluções dadas pelas diferentes abordagens criamos os gráficos da Figura 5.6. Nesses gráficos, o eixo y indica a porcentagem de instâncias que mantiveram a cor de pelo menos $x\%$ da quantidade de vértices que o PLI manteve. Por exemplo, no gráfico da Figura 5.6(a) todas as instâncias mantiveram a cor de pelo menos 80% do número de vértices que o PLI manteve.

No gráfico da Figura 5.6(a) temos os resultados referentes ao conjunto *ratio*, e na Figura 5.6(b) os resultados referentes ao conjunto *union*. Vemos nesses gráficos, que as duas abordagens com a busca local usando a *vizinhança de troca* têm mais soluções com custos próximos daqueles dados pelo modelo PLI modificado.

Como existe uma diferença significativa entre essas abordagens (Linhas 3 e 4 da Tabela 5.1), executamos o teste de Nemenyi para verificar quais versões realmente diferem entre si. Os resultados desses testes são apresentados na Figura 5.7. Os testes indicam que as abordagens que usam busca local com a *vizinhança simples* e a *vizinhança estendida*



(a) ***ratio*** (b) ***union***

Figura 5.6: Comparando as soluções dadas pelo GRASP com o PLI (MRRP). Nos gráficos das Figuras 5.6(a) e 5.6(b), temos a quantidade de instâncias das abordagens GRASP que mantiveram as cores originais (em relação à solução obtida pelo PLI).

são equivalentes, para ambos os conjuntos *ratio* (Figura 5.7(a)) e *union* (Figura 5.7(b)). Entretanto, com os testes individuais, nas Linhas 5 e 6 da Tabela 5.1, temos que apenas as versões da busca local com o critério ratio não têm uma diferença significativa.

Nos gráficos na Figura 5.8, temos a diferença em números de vértices recoloridos das abordagens com o GRASP e do modelo PLI. Um valor positivo no eixo y indica que o GRASP recoloriu mais vértices que o PLI. Vemos que os dois critérios retornaram muitas soluções com o mesmo custo dado pelo modelo PLI. Além disso, o GRASP recoloriu no máximo dois vértices a mais que o modelo.

Ainda com os resultados desse experimento, decidimos comparar os dois critérios na versão que usa a *vizinhança de troca*. Fizemos isto, pois temos resultados muito parecidos nas duas abordagens. O resultado do teste estatístico pode ser encontrado na Linha 7 da Tabela 5.1. Esse teste nos diz que, quando usamos a busca local com a *vizinhança de troca*, os critérios *ratio* e *union* não possuem uma diferença estatisticamente significativa.

De fato, ao contarmos quantas vezes um conjunto foi melhor que outro, encontramos que o *ratio* deu a melhor solução em apenas sete instâncias (de 5000) e o *union* em apenas quatro instâncias. Como os dois conjuntos são equivalentes, não houve ganho significativo em usar uma abordagem que combina os dois conjuntos (que era o caso no versão do GRASP para o CR). Escolhemos usar a versão do GRASP com o conjunto *union* e a *vizinhança de troca* nos experimentos seguintes, pois essa abordagem recoloriu no máximo um vértice a mais que o PLI.

Executamos os procedimentos de pós-processamento especificados na Seção 5.2, contudo nenhuma delas conseguiu melhorar as soluções dada pela abordagem que usa a *vizinhança de troca*.

Por fim, fizemos um teste estatístico comparando o PLI e o GRASP (Linha 8), e não conseguimos mostrar que existe uma diferença estatisticamente significativa entre as duas abordagens.

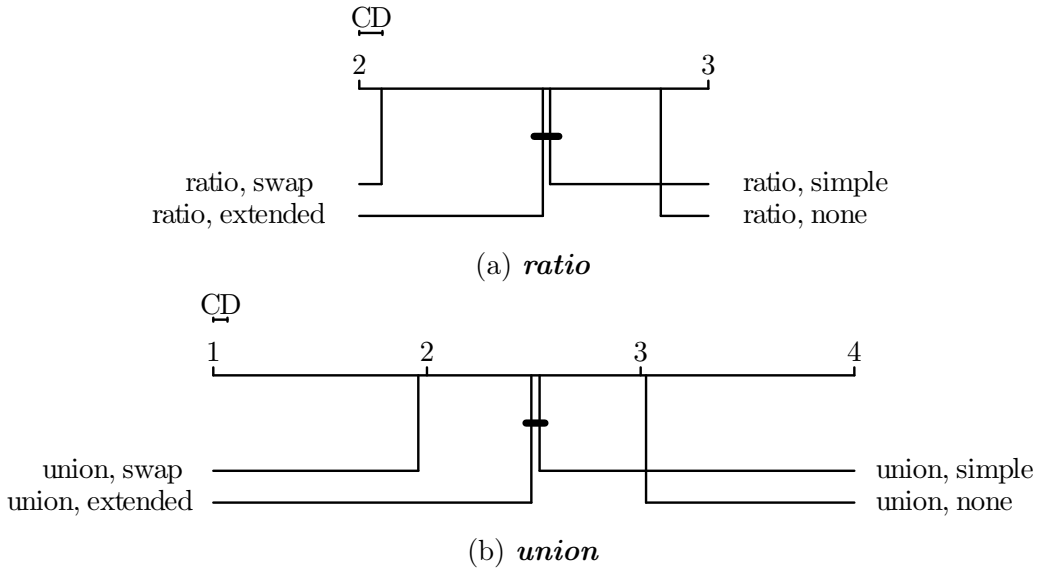


Figura 5.7: Diferença crítica entra as versões do GRASP com busca local. Nesses gráficos, as abordagens que não estão conectadas por uma linha horizontal possuem uma diferença estatisticamente significativa.

5.4 Conclusões

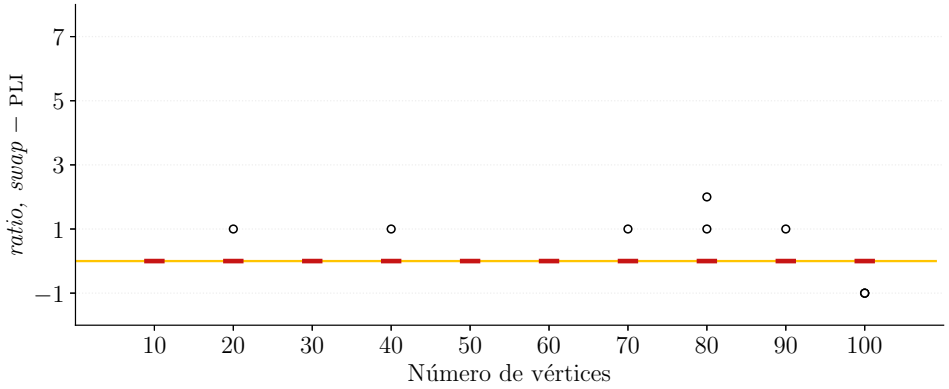
Neste capítulo, adaptamos o modelo de Programação Linear Inteira (PLI) proposto por Campêlo *et al.* [5] e a heurística GRASP proposta no capítulo anterior para tratarem do problema de Recoloração Convexa Restrita (MRRP).

Adicionamos um conjunto de restrições que proíbem o vértice cliente de receber uma cor diferente da sua original. Já na adaptação do GRASP, foram feitas mudanças nas escolhas das novas cores que os vértices recebem e nos critérios de ordenação dos vértices candidatos. Também especificamos conjuntos de vértices servidores para as instâncias do CR, de modo de a transformá-las em instâncias para o MRRP.

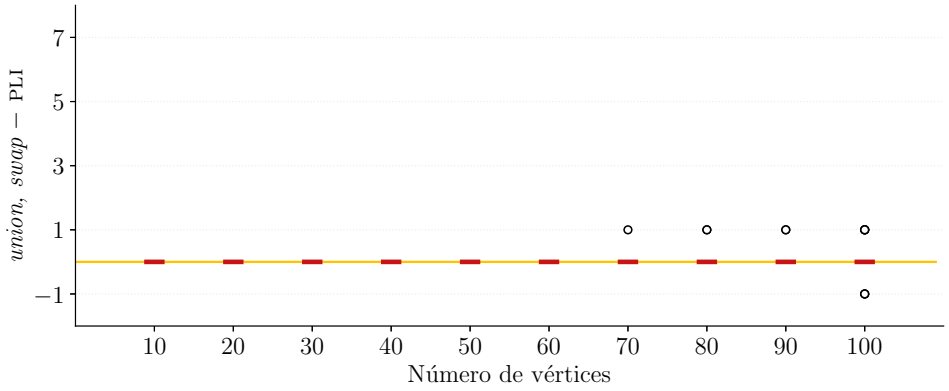
Usamos as implementações do capítulo anterior, adicionando as adaptações. As instâncias adaptadas para o MRRP usam apenas as instâncias com no máximo 100 vértices e valores de p entre $\{0.1, 0.2, 0.3, 0.4, 0.5\}$, totalizando 5000 instâncias.

Com os experimentos reportados neste capítulo, verificamos que os dois conjuntos de critérios gulosos definidos para o GRASP retornaram soluções parecidas e não apresentam diferenças significativas. Além disso, a busca local com a *vizinhança de troca* também foi a vizinhança que deu os melhores resultados. Já os procedimentos de pós-processamento não apresentaram melhoras significativas quando usamos a *vizinhança de troca*. O GRASP retornou soluções muito similares às do PLI, sendo que para 99% das instâncias, o GRASP retornou uma solução com o mesmo custo da solução retornada pelo PLI.

A versão do MRRP tratada neste trabalho considera que toda classe de cor deve induzir um subgrafo com apenas uma componente conexa. Contudo, considerando o contexto de redes de computadores pode ser interessante limitar o número de componentes a um valor k. Isto é, cada classe de cor deve induzir no máximo k componentes. Sendo assim, acreditamos que esta variante possa ser tratada em estudos futuros.



(a) ***ratio***



(b) ***union***

Figura 5.8: Comparando as melhores versões do GRASP com o PLI. Nesses gráficos, um valor positivo no eixo y indica que a versão do GRASP recoloriu mais vértices.

Experimento	**Teste**	**Valor**	**Condição**	**Rejeita**
1. *ratio*: versões puramente gulosa vs aleatorizada	Wilcoxon	$z = -59.48$	$z < -1.96$	Sim
2. *union*: versões puramente gulosa vs aleatorizada	Wilcoxon	$z = -58.69$	$z < -1.96$	Sim
3. *ratio*: vizinhanças para a busca local	Iman-Davenport	$F_F = 348.05$	$F_F > 2.61$	Sim
4. *union*: vizinhanças para a busca local	Iman-Davenport	$F_F = 642.14$	$F_F > 2.61$	Sim
5. *ratio*: *vizinhança estendida* vs *vizinhança simples*	Wilcoxon	$z = -1.36$	$z < -1.96$	Não
6. *union*: *vizinhança estendida* vs *vizinhança simples*	Wilcoxon	$z = -2.69$	$z < -1.96$	Sim
7. *ratio* vs *union*	Wilcoxon	$z = -0.07$	$z < -1.96$	Não
8. *union* vs PLI	Wilcoxon	$z = -0.15$	$z < -1.96$	Não

Tabela 5.1: Resumo dos testes estatísticos (MRRP), considerando um nível de confiança de 0.95. As linhas desta tabela são referenciadas ao longo da seção para indicar a qual experimento os valores dos testes se referem.

Capítulo 6

Considerações Finais

Neste trabalho, estudamos três versões do problema de recoloração convexa de grafos. Inicialmente tratamos da versão com árvores (*Convex Tree Recoloring* – CTR) incluindo o caso particular que admite uma coloração inicial apenas das folhas. Nessa fase do trabalho, que é apresentada no Capítulo 3, replicamos experimentos encontrados na literatura e propomos novas instâncias para o problema com apenas folhas coloridas.

Em seguida, abordamos o problema de recoloração convexa com grafos gerais (*Convex Recoloring* – CR). Para essa versão, implementamos um modelo de Programação Linear Inteira (PLI) da literatura, propusemos uma heurística baseada no GRASP que retornou um número significativo de soluções melhores que as do modelo PLI, ao ser disponibilizada a mesma quantidade de recursos computacionais. Também criamos um conjunto de instâncias para o problema que foi usado nos experimentos reportados no Capítulo 4.

Parte dos resultados do Capítulo 4 foi apresentado no artigo *"A GRASP for the Convex Recoloring Problem"* que foi aceito para apresentação no *"X Latin and American Algorithms, Graphs and Optimization Symposium"* (LAGOS 2019) e subsequente publicação no *Electronic Notes in Theoretical Computer Science* [11].

Por último, tratamos do problema de recoloração convexa restrita (*Minimum Restricted Recoloring Problem* – MRRP), versão na qual um subconjunto de vértices não pode ser colorido com uma cor diferente da sua cor inicial, isto é, podem apenas ser descoloridos. Para essa versão, não há restrições na classe do grafo de entrada, por isso adaptamos para a versão restrita as abordagens apresentadas no Capítulo 4, que trataram do problema de recoloração convexa com grafos gerais. Nos experimentos com o MRRP, reportados no Capítulo 5 a heurística GRASP também obteve bons resultados, recolorindo no máximo um vértice a mais que a melhor solução dada pelo modelo PLI.

Ao final de cada capítulo, apresentamos possíveis trabalhos futuros para cada versão do problema. Para a versão em árvores, uma abordagem interessante seria tratar de árvores filogenéticas que representam mais de uma característica. Isso implica no uso de mais de uma coloração na árvore.

Uma abordagem interessante para a versão em grafos gerais é executar experimentos com outros tipos de grafos de entrada e outras colorações iniciais como, por exemplo, a versão do problema apresentada por Campêlo *et al.* [6]. Nesta versão, o problema é visto como um jogo de tabuleiro que usa moedas com um lado branco e o outro preto. Em cada casa do tabuleiro as moedas são dispostas de forma arbitrária. Um jogador deve

fazer virar o menor número de moedas, de modo que cada cor ocupe uma única região conectada do tabuleiro. Ou seja, temos uma versão do problema de recoloração convexa onde o grafo de entrada é um *grid*, a coloração inicial usa apenas duas cores e a coloração final deve ser total.

Já para a versão restrita do problema de recoloração convexa, uma possível direção de pesquisa é considerar que cada classe de cor induz no máximo k componentes conexas, com $k > 1$.

Referências Bibliográficas

[1] Emgad H. Bachoore and Hans L. Bodlaender. Convex Recoloring of Leaf-Colored Trees. Technical Report UU-CS-2006-010, Department of Information and Computing Sciences, Utrecht University, 2006.

[2] Reuven Bar-Yehuda, Ido Feldman, and Dror Rawitz. Improved Approximation Algorithm for Convex Recoloring of Trees. *Theory of Computing Systems*, 43(1):3–18, 2008.

[3] Reuven Bar-Yehuda, Gilad Kutiel, and Dror Rawitz. 1.5-Approximation Algorithm for the 2-Convex Recoloring Problem. In *Proceedings of the 26th International Workshop on Combinatorial Algorithms (IWOCA'2015)*, volume 9538 of *Theoretical Computer Science and General Issues*, pages 299–311. Springer International Publishing, 2015.

[4] John A. Bondy and Uppaluri S. R. Murty. *Graph Theory*, volume 244 of *Graduate Texts in Mathematics*. Springer-Verlag London, 1st edition, 2011.

[5] Manoel B. Campêlo, Alexandre S. Freire, Karla R. Lima, Phablo F. S. Moura, and Yoshiko Wakabayashi. The Convex Recoloring Problem: Polyhedra, Facets and Computational Experiments. *Mathematical Programming*, 156(1-2):303–330, 2016.

[6] Manoel B. Campêlo, Cristiana G. Huiban, Rudini M. Sampaio, and Yoshiko Wakabayashi. On the Complexity of Solving or Approximating Convex Recoloring Problems. In *Proceedings of the 19th International Computing and Combinatorics Conference (COCOON'2013)*, volume 7936 of *Lecture Notes in Computer Science*, pages 614–625. Springer Berlin Heidelberg, 2013.

[7] Manoel B. Campêlo, Karla R. Lima, Phablo F. S. Moura, and Yoshiko Wakabayashi. Polyhedral Studies on the Convex Recoloring Problem. *Electronic Notes in Discrete Mathematics*, 44:233–238, 2013.

[8] Sunil Chopra, Ergin Erdem, Eunseok Kim, and Sangho Shim. Column Generation Approach to the Convex Recoloring Problem on a Tree. In *Proceedings of the 16th Modeling and Optimization: Theory and Applications Conference (MOPTA'2016)*, volume 213 of *Springer Proceedings in Mathematics & Statistics*, pages 39–53. Springer International Publishing, 2016.

[9] Sunil Chopra, Bartosz Filipecki, Kangbok Lee, Minseok Ryu, Sangho Shim, and Mathieu Van Vyve. An Extended Formulation of the Convex Recoloring Problem on a Tree. *Mathematical Programming*, 165(2):529–548, 2017.

[10] Benny Chor, Michael Fellows, Mark A. Ragan, Igor Razgon, Frances Rosamond, and Sagi Snir. Connected coloring completion for general graphs: algorithms and complexity. In *Proceedings of the 13th Annual International Computing and Combinatorics Conference (COCOON'2007)*, volume 4598 of *Lecture Notes in Computer Science*, pages 75–85. Springer Berlin Heidelberg, 2007.

[11] Ana P. S. Dantas, Cid C. de Souza, and Zanoni Dias. A GRASP for the Convex Recoloring Problem on Graphs, 2019. To Appear in Proceedings of the X Latin and American Algorithms, Graphs and Optimization Symposium Conference (LAGOS 2019).

[12] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[13] Balázs Dezös, Alpár Jüttner, and Péter Kovács. LEMON - an Open Source C++ Graph Template Library. *Electronic Notes in Theoretical Computer Science*, 264(5):23 – 45, 2011.

[14] Thomas A. Feo and Mauricio G. C. Resende. A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. *Operations Research Letters*, 8(2):67 – 71, 1989.

[15] Thomas A. Feo and Mauricio G. C. Resende. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

[16] Thomas A. Feo, Mauricio G. C. Resende, and Stuart H. Smith. A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set. *Operations Research*, 42(5):860–878, 1994.

[17] Thomas A. Feo, Kishore Sarathy, and John McGahan. A GRASP for Single Machine Scheduling with Sequence Dependent Setup Costs and Linear Delay Penalties. *Computers Operations Research*, 23(9):881 – 895, 1996.

[18] Paola Festa and Mauricio G. C. Resende. *GRASP: An Annotated Bibliography*, pages 325–367. Operations Research/Computer Science Interfaces Series. Springer US, 1st edition, 2002.

[19] Zeev Frenkel, Yosef Kiat, Ido Izhaki, and Sagi Snir. Convex Recoloring as an Evolutionary Marker. *Molecular Phylogenetics and Evolution*, 107:209–220, 2017.

[20] Andrew V. Goldberg and Robert E. Tarjan. A New Approach to the Maximum-flow Problem. *Journal of the ACM*, 35(4):921–940, 1988.

[21] Leslie Ann Goldberg, Paul W. Goldberg, Cynthia A. Phillips, Elizabeth Sweedyk, and Tandy Warnow. Minimizing phylogenetic number to find good evolutionary trees. In *Proceedings of the 6th Annual Symposium Combinatorial Pattern Matching (CPM'1995)*, volume 937 of *Lecture Notes in Computer Science*, pages 102–127. Springer Berlin Heidelberg, 1995.

[22] Frank Kammer and Torsten Tholey. The Complexity of Minimum Convex Coloring. *Discrete Applied Mathematics*, 160(6):810–833, 2012.

[23] Iyad A. Kanj and Dieter Kratsch. Convex Recoloring Revisited: Complexity and Exact Algorithms. In *Proceedings of the 15th International Computing and Combinatorics Conference (COCOON'2009)*, volume 5609 of *Lecture Notes in Computer Science*, pages 388–397. Springer Berlin Heidelberg, 2009.

[24] Manuel Laguna and Rafael Martí. A GRASP for Coloring Sparse Graphs. *Computational Optimization and Applications*, 19(2):165–178, 2001.

[25] Karla R. Lima. *Recoloração Convexa de Caminhos*. PhD thesis, Instituto de Matemática e Estatística, Universidade de São Paulo, 2011.

[26] Karla R. Lima and Yoshiko Wakabayashi. Convex Recoloring of Paths. *Discrete Applied Mathematics*, 164:450–459, 2014.

[27] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Thomas Stützle, and Mauro Birattari. The irace package: Iterated Racing for Automatic Algorithm Configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[28] Shlomo Moran and Sagi Snir. Efficient Approximation of Convex Recolorings. *Journal of Computer and System Sciences*, 73(7):1078–1089, 2007.

[29] Shlomo Moran and Sagi Snir. Convex Recolorings of Strings and Trees: Definitions, Hardness Results and Algorithms. *Journal of Computer and System Sciences*, 74(5):850–869, 2008.

[30] Shlomo Moran, Sagi Snir, and Wing-Kin Sung. Partial Convex Recolorings of Trees and Galled Networks: Tight Upper and Lower Bounds. *ACM Transactions on Algorithms*, 7(4):42, 2011.

[31] Phablo F. S. Moura. *Graph Colorings and Digraph Subdivisions*. PhD thesis, Instituto de Matemática e Estatística, Universidade de São Paulo, 2017.

[32] Oriana Ponta, Falk Hüffner, and Rolf Niedermeier. Speeding up Dynamic Programming for Some NP-Hard Graph Recoloring Problems. In *Proceedings of the 5th International Conference on Theory and Applications of Models of Computation (TAMC'2008)*, volume 4978 of *Lecture Notes in Computer Science*, pages 490–501. Springer Berlin Heidelberg, 2008.

[33] Mauricio G. C. Resende and Celso C. Ribeiro. GRASP with Path-Relinking: Recent Advances and Applications. In *Metaheuristics: Progress as Real Problem Solvers*, pages 29–63. Springer US, 2005.