Problema da coloração de vértices com pesos dissonantes e restrições de cores

Edison Gabriel Gonçalves Borghezan

SERVIÇO DE PÓS-GRADUAÇÃO DA FACOM-UFMS

Data de Depósito:

Assinatura:_____

Problema da coloração de vértices $_5$ com pesos dissonantes e restrições $_5$ de cores

Edison Gabriel Gonçalves Borghezan

Orientadora: *Profª. Drª. Edna Ayako Hoshino*

Coorientador: *Prof. Dr. Vagner Pedrotti*

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação no Programa de Pós-graduação em Ciência da Computação da Universidade Federal de Mato Grosso do Sul.

UFMS - Campo Grande
julho/2023

Abstract

The vertex coloring problem with dissonant weights and color constraints is a generalization of the vertex coloring problem and several other coloring problems can be reduced to it. This work proposes a variation of the vertex coloring problem, and also three mathematical models using integer linear programming, a model whose number of variables and restrictions is polynomial, and a second, in which the number of variables is exponential in relation to the number of restrictions and a third model very similar to the second but which takes advantage of a property that allows some colors to be combined for faster execution. For the extended models, column generation algorithms are proposed to deal with the exponential number of variables in the problem, as well as heuristic, both to generate new columns and to look for integer solutions on each node of the enumeration tree to accelerate the performance of the exact branch-and-price algorithm. A set of instances was proposed and it was possible to identify characteristics of the instances that are the most difficult ones for the problem.

Resumo

O problema de coloração de vértices com pesos dissonantes e restrições de cores é uma generalização do problema de coloração de vértices e vários outros problemas de coloração podem ser reduzidos a ele. Neste trabalho é proposta uma variação do problema de coloração de vértices, e também três modelos matemáticos utilizando programação linear inteira, um modelo cujo número de variáveis e restrições é polinomial, um segundo, no qual o número de variáveis é exponencial em relação ao número de restrições e um terceiro modelo bastante semelhante ao segundo mas que aproveita-se de uma propriedade que permite algumas cores serem aglutinadas almejando uma execução mais rápida. Para os modelos estendidos, são propostos algoritmos de geração de colunas para lidar com o número exponencial de variáveis do problema, assim como heurísticas, tanto para gerar novas colunas quanto para encontrar soluções inteiras em cada nó da árvore de enumeração para acelerar o desempenho de um algoritmo exato de *branch-and-price*. Um conjunto de instâncias foi proposto e foi possível identificar características das instâncias difíceis para este problema.

Sumário

CAPÍTULO

1

Introdução

Recorrentemente, em situações práticas, surge a necessidade de agrupar elementos de acordo com algum critério de compatibilidade.

Considere, por exemplo, que em um laboratório de química, os reagentes químicos precisam ser organizados em prateleiras. No entanto, alguns reagentes são incompatíveis, ou seja, caso estejam juntos podem ocasionar acidentes. Portanto, deseja-se separá-los em prateleiras de forma que nenhuma prateleira possua algum par incompatível de reagentes e que seja utilizado o menor número possível de prateleiras.

Se considerarmos um grafo, no qual o conjunto de vértice é o conjunto de reagentes e para cada par incompatível de reagentes existe uma aresta ligando os vértices correspondentes, resolver o problema da organização dos reagentes é o mesmo que resolver o problema de coloração de vértices neste grafo.

O problema alvo, que é o objeto de estudo desta proposta de dissertação de mestrado, é uma variação nova do problema de coloração de vértices e possui aplicações em problemas de atribuição de tarefas a máquinas com restrições de incompatibilidade. Nessas aplicações, há uma incompatibilidade entre tarefas, dada por um grafo de conflitos, no qual cada vértice do grafo representa uma tarefa e cada aresta (i, j) representa a incompatibilidade de atribuir as tarefas i e j a uma mesma máquina. Adicionalmente, há um peso associado a cada máquina e indica o custo de configuração e ativação da máquina. Cada tarefa tem um custo de atribuição, dependendo da máquina à qual é ela atribuída e, também, existe uma lista de máquinas compatíveis para as quais uma tarefa pode ser atribuída. O objetivo nessas aplicações é atribuir com o menor custo possível todas as tarefas às máquinas, considerando-se os custos de ativação das máquinas e o custo da atribuição de cada uma das

tarefas, respeitando as restrições de compatibilidade entre tarefas e máquinas.

Na próxima seção são apresentadas algumas definições e notações que são essenciais para a compreensão do problema.

1.1 Notações e definições gerais

As notações usadas nesta dissertação são consistentes com os livros Bondy e Murty (2008) e Gross e Yellen (2004).

Um **grafo** G é um par ordenado $(V(G),E(G))$ consistindo em um conjunto $V(G)$ de vértices e um conjunto $E(G)$ de arestas disjunto de $V(G)$, juntamente com uma função de incidência ψ_G que associa a cada aresta de G um par de vértices (não necessariamente distintos) de G.

Seja e uma aresta, u e v vértices tais que $\psi_G(e) = \{u,v\}$, diz-se que e é uma **ligação** entre u e v, os vértices u e v são **extremidades** de e.

Um vértice u é **adjacente** a um vértice v se eles estão ligados por uma aresta. Dois vértices adjacentes também são chamados de **vizinhos**. Usaremos também as notações n para representar $|V(G)|$ e m para $|E(G)|$, ou seja, as cardinalidades de $V(G)$ e de $E(G)$, respectivamente. Uma aresta com extremidades iguais é chamada de **laço**, aresta com extremidades diferentes é chamada de ligação ou **adjacência** e duas ou mais ligações com os mesmos pares de vértices são ditas **arestas paralelas**. Um **grafo simples** é um grafo que não possui laços nem arestas paralelas. Um **grafo completo** é um grafo simples tal que para todo par de vértices distintos existe adjacência. Um **subgrafo** de um grafo G é um grafo H tal que $V_H \subseteq V_G$ e $E_H \subseteq E_G$. Um **subgrafo induzido** por um conjunto de vértices Y de um grafo G, denotado por $G[Y]$ é um subgrafo de G, no qual o conjunto de vértices é Y e o conjunto de arestas consiste em todas as arestas de G que possuem as duas extremidades em Y.

Uma **k-coloração** de um grafo é a atribuição de k cores para seus vértices. Também pode ser descrita como uma função $f : V \to \mathcal{C}$, na qual $\mathcal{C}$ é um conjunto de cardinalidade k cujos elementos são denominados cores.

Um subconjunto de vértices no qual não existem dois vértices adjacentes é chamado de **conjunto independente**.

Uma k-coloração é equivalente a particionar o conjunto V de vértices em k conjuntos $V_1,V_2,\ldots,V_k$. Se cada grafo induzido $G[V_i]$, para todo $i \in \{1,2,...,k\}$, for um conjunto independente, é dito que f é uma k-**coloração própria**.

Uma coloração em um grafo G é **própria** se não existirem dois vértices adjacentes em G que estejam associados a uma mesma cor na coloração.

Um grafo G é k-**colorável** se existir uma k-coloração própria para G.

Dada uma função $g : A \to B$ e $X \subset A$, a **imagem direta** de X por g, denotada por $g(X)$, é o subconjunto $g(X) = \{g(x)|x \in X\}$ de B.

Dado um conjunto A, o **conjunto das partes** de A, denotado por $P(A)$, é o conjunto $P(A) = \{x | x \subseteq A\}$.

Para facilitar a leitura, utilizaremos $\mathbb{B}$ para denotar o conjunto $\{0,1\}$.

1.2 Problemas de coloração

A fim de facilitar a compreensão do problema alvo serão apresentados outros dois problemas adicionais de coloração de vértices.

1.2.1 Problema clássico de coloração de vértices

Dado um grafo G, **o Problema da Coloração de Vértices**, do inglês, *Vertex Coloring Problem* (**VCP**) consiste em encontrar o menor inteiro k tal que G seja k-colorável. Seja o grafo exposto na Figura 1.1, observa-se uma 2-coloração. É possível pintar o vértice rotulado por v de uma terceira cor, porém, no problema da coloração de vértices clássico deseja-se utilizar o menor número de cores possível. Dessa forma, não é possível utilizar apenas uma cor para colorir este grafo, visto que ele possui arestas. Observe que esta é apenas uma das soluções.

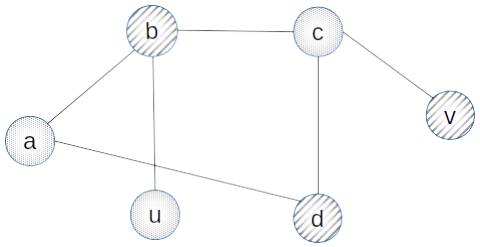

Figura 1.1: Exemplo de coloração de vértices clássico.

1.2.2 Problema da coloração de vértices de soma mínima

Dado um grafo G e uma função $c : \mathcal{C} \to \mathbb{R}$, denominada função de custo da cor, no **problema da coloração de vértices de soma mínima**, do inglês, *Minimal sum coloring problem* (MSCP) deseja-se encontrar uma coloração própria tal que $\sum_{v \in V} c(f(v))$ seja mínima.

Nas Figuras 1.2 e 1.3, considere que o custo do padrão hachurada na diagonal é um, o custo do padrão na vertical é dois e o custo do padrão pontilhado é três. O peso de cada cor também é apresentado nas figuras.

Na Figura 1.2 é exposto um grafo colorido de forma ótima considerando-se o problema clássico da coloração. Considerando-se a função custo descrita anteriormente, tem-se que o valor associado a essa coloração é 12. No entanto, observe que na Figura 1.3, o mesmo grafo foi colorido utilizando-se mais cores,

porém na coloração de vértices com peso, esta é a coloração ótima, cujo custo de coloração tem valor igual a 11.

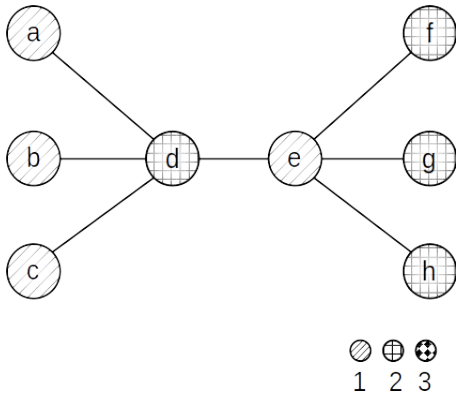

Figura 1.2: Exemplo de coloração de vértices com pesos utilizando a estratégia clássica.

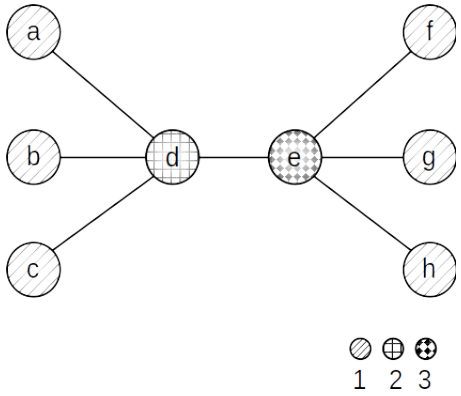

Figura 1.3: Exemplo de coloração de vértices com pesos.

O exemplo anterior deixa claro que uma coloração ótima para um grafo no problema clássico da coloração de vértices nem sempre é uma coloração ótima para o grafo no WVCP.

1.2.3 *Problema da coloração de vértices com pesos dissonantes e restrições de cores*

Dado um grafo G, um conjunto $\mathcal{C}$ de cores, uma função $w : V \times \mathcal{C} \to \mathbb{R}$, denominada função de custo de coloração do vértice, uma função de custo da cor $c : \mathcal{C} \to \mathbb{R}$ e uma função $l : V \to \mathcal{P}(\mathcal{C})$, denominada restrição de lista de cores, **o problema da coloração de vértices com pesos dissonantes e restrições de cores** (VCPDWC) consiste em encontrar uma coloração própria de G que minimize $\sum_{v \in V} w(v, f(v)) + \sum_{k \in f(V)} c(k)$ tal que $f(v) \in l(v)$, para todo $v \in V$.

Considere que a função de custo c atribua custo um à cor representada pelo padrão hachurado na diagonal, custo dois à cor representada pelo padrão na vertical e custo três à cor representada pelo padrão pontilhado. Nas

Figuras 1.4 e 1.5, os custos de coloração do vértice estão indicados em cada um dos vértices. Um vértice que possui custo de coloração ∞ para uma cor representa que aquela cor não pode ser utilizada naquele vértice, isto é, aquela cor não pertence à restrição de lista de cores do vértice.

A Figura 1.4 apresenta uma coloração ótima do grafo, considerando-se o problema clássico de coloração de vértices. Calculando-se o valor desta solução, considerando-se os custos de coloração no VCPWDC, obteve-se o valor 18. Na Figura 1.5, o mesmo grafo foi colorido utilizando-se mais cores que a coloração clássica. Porém, o valor associado à coloração, considerando-se os custos no VCPWDC, é igual a 16, ou seja, a coloração que utiliza mais cores apresentou um custo de coloração menor.

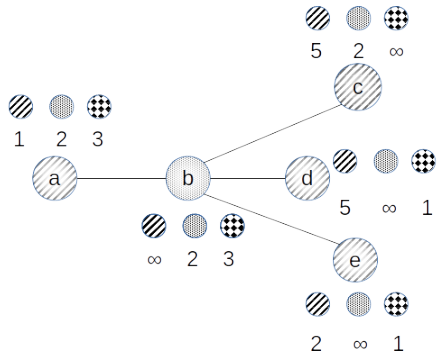

Figura 1.4: Exemplo de coloração de vértices com pesos dissonantes e restrições de cores, utilizando a estratégia clássica.

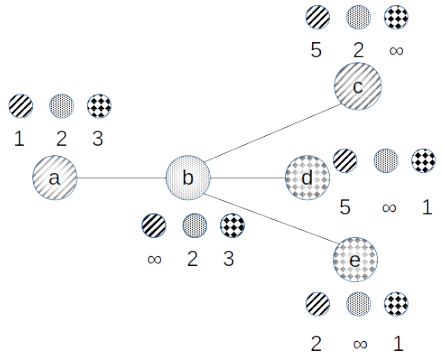

Figura 1.5: Exemplo de coloração de vértices com pesos dissonantes e restrições de cores.

Novamente, os exemplos ilustram um caso em que a coloração clássica de vértices não resolve a variante estudada.

1.3 Trabalhos relacionados

Nesta seção são apresentados alguns problemas relacionados ao VCPWDC. Uma vez que, trata-se de um problema novo, não existem trabalhos na litera-

tura que discutem abordagens para ele. Portanto, escolhemos alguns problemas semelhantes para inspirar o nosso trabalho.

Regra de branching same-diff

A regra de branching same-diff, proposta por Zykov (1949) é a regra de *branching* mais utilizada em problemas de coloração de vértices. Considere o grafo descrito na Figura 1.6.

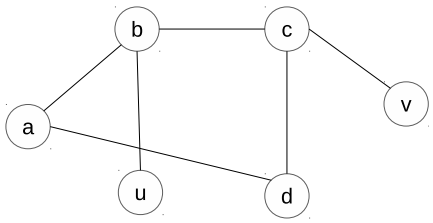

Figura 1.6: Grafo exemplo para regra de *branching*.

No problema de coloração de vértices, como u e v não são adjacentes, temos duas possibilidades para uma solução:

$$f(u) = f(v) \text{ ou } f(u) \neq f(v).$$

Portanto, pode-se dividir o espaço de soluções em dois, um no qual $f(u) = f(v)$ e a outro em que $f(u) \neq f(v)$. Para fazer isso, podemos alterar o grafo de entrada para cada um dos ramos.

No primeiro caso, quando queremos que $f(u) = f(v)$, criaremos um novo vértice w que é adjacente a todos os vértices adjacentes a u e a v e removemos os vértices u e v do grafo. Na Figura 1.7 é mostrado o grafo alterado para o ramo em que $f(u) = f(v)$.

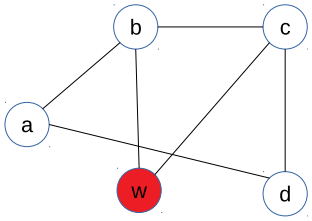

Figura 1.7: Grafo da Figura 1.6 alterado pela aplicação da regra de *branching* no ramo em que $f(u) = f(v)$.

No segundo caso, para garantir que $f(u) \neq f(v)$, basta adicionar a aresta $e = \{u, v\}$ ao grafo de entrada. A Figura 1.8 apresenta o grafo alterado para o ramo em que $f(u) \neq f(v)$.

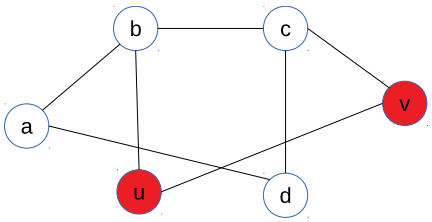

Figura 1.8: Grafo da Figura 1.6 alterado pela aplicação da regra de *branching* no ramo em que $f(u) \neq f(v)$.

Problema de coloração de vértices com pesos

Dado um grafo $G = (V, E)$ e $w : V \to \mathbb{R}$, o Problema de coloração de vértices com pesos, do inglês, *Weighted Vertex Coloring Problem* (**WVCP**) consiste em encontrar uma coloração própria f, na qual S é a partição de V definida por f e de forma que $\sum_{s \in S} \max_{v \in s} w(v)$ seja mínima.

No artigo de Furini e Malaguti (2012), o WVCP é resolvido via *branch-and-price*, utilizando a regra de *branching same-diff*. Foi proposta pelos autores uma abordagem para selecionar o par de vértices para o *branching* baseada na diferença dos pesos dos vértices, buscando selecionar prioritariamente os pares com menor diferença.

Minimal sum coloring problem

O problema aqui referido é o mesmo já descrito na Seção 1.2.2. Na tese de Ternier (2017) o MSCP é resolvido de forma exata utilizando um algoritmo de *branch-and-price*.

No artigo de Wu e Hao (2012) é proposta uma heurística para o MSCP, chamada EXSCOL, baseada na extração de conjuntos independentes. A heurística consiste na identificação iterativa de múltiplos conjuntos independentes disjuntos de mesmo tamanho e associa a cada conjunto independente a menor cor disponível. Para a identificação dos conjuntos independentes utiliza-se uma heurística baseada em Busca Tabu. Em mais da metade dos grafos comumente utilizados na literatura, o EXSCOL melhorou o limitante superior conhecido.

O MSCP, definido na Seção 1.2.2, tem uma forte relação com o problema estudado nesta dissertação. Uma instância do MSCP pode ser reduzida a uma instância do VCPDWC em que os pesos de utilizar uma cor sempre é 0 ($c(k) = 0 \ \forall k \in \mathcal{C}$), os pesos para colorir os vértices não são dissonantes, ou seja, o custo de colorir qualquer vértice com uma certa cor k é igual para todo vértice ($w(v,k) = w(k) \ \forall v \in V, \ \forall k \in \mathcal{C}$) e não existem restrições de cores que podem ser utilizadas, ou seja, todas as cores são permitidas em todos os vértices ($l(v) = \mathcal{C} \ \forall v \in V$).

1.4 Objetivo do trabalho e organização do texto

Geralmente, em problemas de coloração de vértices os modelos compactos, ou seja, com um número de variáveis polinomial no tamanho da instância, possuem um limitante ruim para a relaxação linear. Um dos métodos mais comuns para resolver um problema de coloração é um algoritmo de *branch-and-price*, no qual resolve-se a relaxação linear de um modelo estendido, no qual o número de variáveis é exponencial no tamanho da instância. Para resolver a relaxação linear nestes modelos estendidos, utiliza-se o método da geração de colunas.

Este trabalho propõe três modelos de programação linear para o VCPDWC, um compacto e dois estendidos, um algoritmo exato de *branch-and-price* baseados nos modelos estendidos e heurísticas para reduzir o tempo de execução.

No Capítulo 2 são apresentadas algumas notações, definições e alguns métodos necessários para a compreensão do trabalho apresentado no Capítulo 3. No Capítulo 4 são apresentados os experimentos computacionais e no Capítulo 5 as conclusões.

CAPÍTULO

2

Metodologia

Neste capítulo são apresentados os principais conceitos e notações, bem como os métodos utilizados, os quais são importantes para a compreensão do trabalho desenvolvido. Este capitulo foi baseado nos livros de Bazaraa e Jarvis (1977) e Wolsey (1998).

2.1 Programação linear

Um problema de **Programação Linear** (PL) é um problema de minimização ou maximização de uma função linear definida para um conjunto de variáveis, chamadas **variáveis de decisão**, as quais estão sujeitas a um conjunto de restrições lineares que podem ser equações ou inequações.

Usaremos a notação n para o número de variáveis associadas ao problema e m para o número de restrições do problema.

Portanto um problema de programação linear é descrito como:

$$\max(\text{ou } \min)\ C(x) : \mathbb{R}^n \to \mathbb{R}$$

sujeito às restrições $R_1, R_2, R_3, ..., R_m$, que podem ser escritas como:

$$R_i = \begin{cases} r_i(x) \leq b_i \\ r_i(x) \geq b_i \\ r_i(x) = b_i. \end{cases}$$

A função $C(x)$ pode ser escrita na forma:

$$C(x) = cx + d,\ c \in \mathbb{R}^n, d \in \mathbb{R}.$$

Como estamos interessados em minimizar ou maximizar $c(x)$ e a constante d não altera a solução, consideraremos $C(x)$ da forma:

$$C(x) = cx,\ c \in \mathbb{R}^n.$$

Todo problema de maximização pode ser escrito como um problema de minimização, pois:

$$\max(C(x)) = \max(cx) = -min((-c)x) = -min(c(-x)) = -min(C(-x)).$$

Logo, podemos trabalhar apenas com problemas de minimização.

Toda inequação linear pode ser escrita como uma equação. Para tanto, considere:

$$\begin{aligned} r_i(x) &\leq b_i \\ a_i x + t &\leq b_i \\ a_i x &\leq b_i - t. \end{aligned}$$

Considere uma nova variável nova y_i. Desta forma, podemos reescrever a inequação anterior como uma equação da seguinte forma:

$$a_i x + y_i = b_i - t.$$

Para isso, é necessário que seja adicionada uma restrição de não-negatividade para y_i, ou seja, $y_i \geq 0$.

De forma semelhante também podemos transformar uma inequação do tipo maior igual em uma equação:

$$\begin{aligned} r_i(x) &\geq b_i \\ s_i x + t &\geq b_i \\ s_i x &\geq b_i - t \\ s_i x - y_i &= b_i - t. \end{aligned}$$

Novamente, considere uma variável nova y_i, com a restrição de não-negatividade $y_i \geq 0$.

Logo, podemos escrever, sem perda de generalidade, um problema de programação linear como:

$$z = \min cx$$

sujeito a

$$\begin{aligned} r_1 x &= b_1, \\ r_2 x &= b_2, \\ r_3 x &= b_3, \\ &\vdots \\ r_m x &= b_m \\ x &\geq 0. \end{aligned}$$

Se considerarmos $A = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_m \end{bmatrix}$ e $b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_m \end{bmatrix}$, podemos escrever o problema como :

$$z = \min cx$$

sujeito a

$$\begin{aligned} Ax &= b, \\ x &\geq 0 \end{aligned}$$

Esta forma costuma ser referenciada como forma padrão.

Programação Linear Inteira

Um problema de **programação linear inteira** (PLI) é um problema de programação linear com uma restrição adicional, chamada **restrição de integralidade**, que obriga o valor de cada variável a ser um inteiro.

2.2 Dualidade

Todo problema de programação linear possui um problema de programação linear associado denominado problema **dual**, que possui algumas propriedades que são úteis para resolver o problema original.

A partir deste capítulo, chamaremos o problema principal de problema **primal**. Se o problema primal é um problema de maximização, o problema dual associado é um problema de minimização. Se o problema primal é um problema de minimização, então o problema dual associado é um problema de maximização.

Dada uma solução x_0 de um problema primal, sempre é possível encontrar os valores das variáveis w_0 do problema dual utilizando x_0. Note que w_0 nem sempre é uma solução do problema dual.

Seja cx o valor de uma solução de um problema primal de minimização e cx^* o valor de sua solução ótima, assim como gw o valor da solução do problema dual e gw^* o valor da solução ótima do problema dual. Temos que:

$$cx \geq gw$$
$$cx^* = gw^*.$$

Seja x' uma solução do problema primal e w' os valores das variáveis do problema dual obtidas por x'. Se w' é viável no problema dual, temos que x' é solução ótima do problema primal.

2.3 Custo reduzido

Caso o problema primal possua alguma solução não ótima, o problema dual associado não é viável com esta solução, ou seja, alguma restrição não está sendo respeitada. Assim, de forma a obter a solução ótima, podemos buscar a viabilidade do problema dual, mantendo a viabilidade do problema primal. Cada uma das variáveis do problema primal está associada a alguma restrição do problema dual. Podemos calcular o quão longe cada uma dessas restrições está de ser resolvida na igualdade e este valor é denominado custo reduzido, tendo impacto direto na solução do problema primal.

Quando uma restrição do problema dual está sendo violada em um problema de minimização, o valor do custo reduzido da variável correspondente no problema primal é negativo, caso contrário, esse valor é positivo ou zero.

2.4 Método simplex

A Figura 2.1 apresenta um diagrama que representa o funcionamento do método simplex.

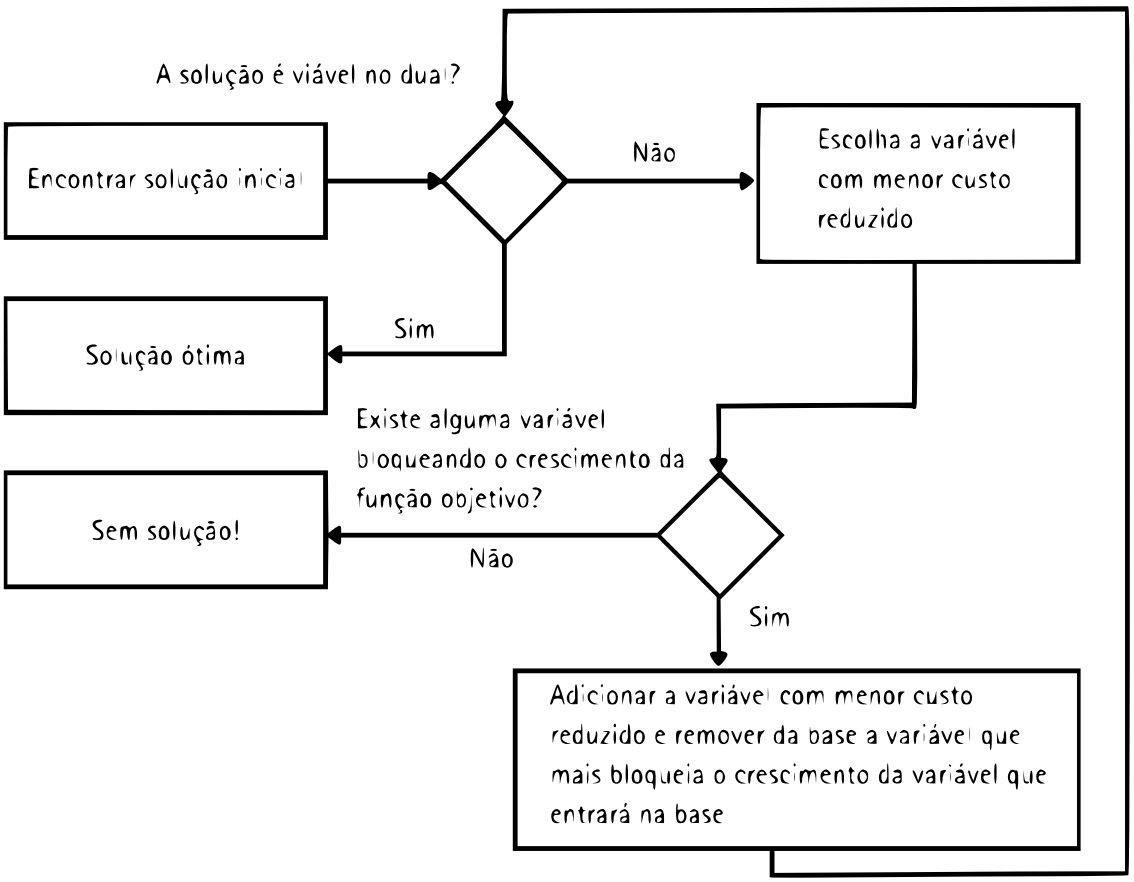

Figura 2.1: Diagrama da execução do algoritmo simplex.

O primeiro passo do algoritmo é encontrar uma solução qualquer do problema, chamada de solução básica. A solução básica é uma solução em que $n-m$ variáveis são fixadas em zero (essas variáveis são chamadas **variáveis não-basicas**) e as colunas das demais variáveis, chamadas **variáveis básicas**, formam uma matriz inversível (chamada **base**). No segundo passo, verificamos se existe alguma variável não-básica com custo reduzido negativo. Se não existir já encontramos a solução ótima, se existir isto quer dizer que uma restrição do dual não está sendo respeitada, então adicionamos a coluna da variável associada a esta restrição na base. Depois devemos escolher uma variável básica de modo que a coluna correspondente a ela seja removida da base. Em geral, seleciona-se a variável básica que está bloqueando mais o crescimento da variável a entrar na base. Caso não exista nenhuma variável impedindo o crescimento da variável a entrar, o problema é ilimitado.

2.5 Geração de colunas

Quando temos um número de variáveis exponencial em relação ao tamanho da instância, é impraticável calcular o custo reduzido de cada uma das variáveis do problema. Para resolver este problema iremos calcular implicitamente o custo reduzido das variáveis do problema. Como procuramos apenas a va-

riável com menor custo reduzido, iremos usar um subproblema que gera uma variável de menor custo reduzido. Em geral é utilizada a nomenclatura Problema Mestre (**PM**) para o problema original, Problema Mestre Restrito (**PMR**) para o problema mestre considerando um conjunto reduzido de variáveis e Problema de *Pricing* (**PP**) para o subproblema que gera uma coluna de menor custo reduzido.

Na Figura 2.2 é apresentado o *framework* para o método de geração de colunas.

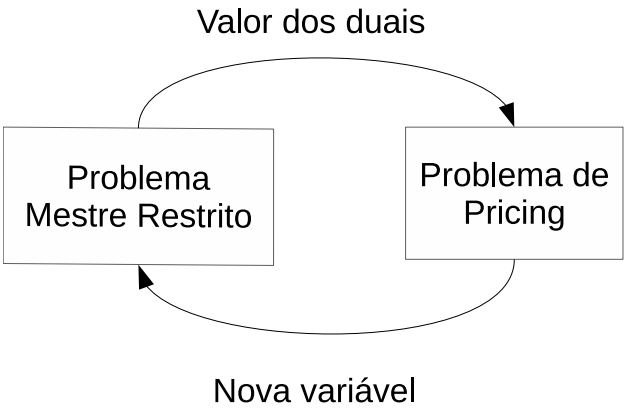

Figura 2.2: Diagrama representando o funcionando de um algoritmo de geração de colunas.

O processo de geração de colunas é iniciado resolvendo-se o PMR, o qual é usualmente formado por um conjunto de variáveis contendo alguma solução viável do problema. A partir da solução ótima do PMR, calcula-se os valores das variáveis duais associadas ao PMR. Após isso, é necessário resolver o PP para gerar uma variável. Para saber se esta nova variável pode ser adicionada ao PMR, calculamos o seu custo reduzido. Se este custo for negativo, a variável é adicionada ao PMR e voltamos ao passo inicial, caso contrário, temos que não existe nenhuma restrição do problema dual sendo violada por nenhuma variável que poderia ser adicionada ao PMR, ou seja, podemos parar o processo pois estamos em uma solução ótima do PM. Este processo é repetido até que não existam variáveis de custo reduzido negativo para serem adicionadas ao PMR. Podemos resolver o PP por meio de heurísticas para que esse passo seja menos custoso computacionalmente, mas sempre que a heurística falhar para encontrar uma coluna de custo reduzido negativo, um algoritmo exato para resolver o PP deve ser usado.

2.6 Branch-and-Bound

Em geral quando trabalhamos com programação linear inteira estamos interessados em problemas NP-difíceis. Em geral, estes problemas são resolvi-

dos utilizando-se o algoritmo *branch-and-bound*.

Para isso, em geral, desconsideram-se as restrições de integralidade, para obter um problema de programação linear, chamado **relaxação linear**, o qual usualmente é resolvido utilizando-se o algoritmo Simplex. Posteriormente, o espaço de solução é dividido em vários subconjuntos do espaço de solução do problema original e repete-se isto em cada um destes subespaços criados. Pode-se representar isto como uma árvore de enumeração, na qual cada nó é um subespaço do espaço de soluções do problema e os filhos de um nó são as divisões do espaço de soluções do nó em questão. Uma árvore de enumeração possui tamanho exponencial, portanto visitar todos os nós exige um tempo de execução muito grande. Para lidar com essa situação, a ideia é verificar, antes de visitar os filhos de cada nó, se existe a possibilidade de encontrar alguma solução melhor naquele ramo da árvore. Caso isso não seja possível, o nó correspondente é desconsiderado. A essa operação dá-se o nome de **poda** da árvore de enumeração.

Um algoritmo de *branch-and-bound* tem como base duas operações básicas: a divisão do espaço de soluções (*branching*) e o cálculo de limitantes (*bound*) para realizar as podas.

Branching

Em cada um dos nós, o espaço de soluções é dividido em dois ou mais subespaços, sendo que, em cada uma das divisões cria-se outro nó. Pode-se dividir este espaço de soluções S de qualquer maneira em n subespaços $S_1, S_2, \ldots, S_n$, desde que:

$$S_1 \cup S_2 \cup \ldots \cup S_n = S.$$

Note que os conjuntos não precisam ser disjuntos. É possível fazer isso fixando uma variável ou criando restrições adicionais.

Limitantes

Para que possamos podar a árvore de enumeração, é preciso que seja calculado um limitante. Cada vez que resolvemos a relaxação linear do problema, existe a possibilidade que esta solução também seja do PLI original, ou seja, aquela solução é um limitante inferior (no caso de problemas de maximização) ou superior (para problemas de minimização) para a solução ótima. Sempre guardamos o maior limitante inferior ou menor limitante inferior, respectivamente.

Podas

- Poda por otimalidade: caso a solução da relaxação linear também é uma solução inteira, não é preciso explorar o nó, já que os filhos deste nó nunca terão solução superior a esta.
- Poda por inviabilidade: se não for possível encontrar nenhuma solução em um nó, não é preciso explorar o nó, pois o conjunto de soluções do nó atual é vazio, logo os possíveis nós filhos possuirão o conjunto de soluções vazios também.
- Poda por limitante: quando o valor da relaxação linear é inferior ao limitante inferior (ou superior ao limite superior, no caso de um problema de minimização), isso quer dizer que todos os possíveis filhos daquele nó também terão valores de solução inferiores ao limitante inferior do problema, ou seja, podemos garantir que a solução ótima não está neste nó, pois já temos uma solução melhor que a máxima do nó.

2.7 Branch-and-Price

Um algoritmo de *branch-and-price* é a utilização das técnicas *branch-and-bound* e geração de colunas em conjunto. No *branch-and-price*, a relaxação linear em cada nó da árvore de enumeração é resolvida usando-se o método da geração de colunas. Em geral, os limitantes duais dados pela relaxação linear de um modelo estendido são mais apertados que os fornecidos pelos modelos compactos. Essa característica contribui para a poda precoce de mais nós da árvore de enumeração.

CAPÍTULO

3

Trabalho Desenvolvido

Este capítulo descreve o trabalho desenvolvido. Ele apresenta três formulações de PLI que foram propostas para modelar o VCPDWC, sendo um modelo compacto e dois modelos estendidos. O primeiro modelo estendido é baseado na formulação de cobertura para o problema clássico de coloração enquanto o segundo aplica uma redução no modelo usando o conceito de cores equivalentes. Para os modelos estendidos, é descrito o problema de *pricing* associado e uma formulação de PLI para modelá-lo. Em seguida, é discutida a regra de *branching* utilizada em um algoritmo *branch-and-price*. Por fim, são apresentadas duas heurísticas para melhorar o desempenho do algoritmo *branch-and-price*, uma para encontrar soluções primais e a outra para gerar colunas.

3.1 Modelos matemáticos para o VCPDWC

O primeiro modelo matemático proposto consiste num modelo compacto muito parecido com o modelo para o problema clássico de coloração de vértices. Este modelo foi inspirado pelo modelo proposto por Lucci et al. (2019) para o problema da coloração de peso mínimo com lista de cores, do inglês *Minimum Weighted List Coloring Problem* (MWLCP). O segundo modelo matemático é um modelo estendido utilizando uma abordagem comum de coloração de vértices, uma formulação que utiliza conjuntos independentes.

A partir deste ponto utilizaremos $V(a) = \{v : v \in V, a \in l(v)\}, \forall a \in \mathcal{C}$ como notação para os vértices de um grafo que podem ser pintados com uma cor a.

3.1.1 Formulação compacta para o VCPDWC

Neste modelo foram utilizados dois conjuntos de variáveis de decisão binárias para representar a solução. As variáveis y_k assumem valor 1 se e somente se a cor k é utilizada na coloração e as variáveis $x_{v,k}$ são iguais a 1 se e somente se o vértice v é colorido com a cor k na solução.

O modelo para o VCPDWC é apresentado a seguir.

$$(F1) \quad z = \min \sum_{v \in V} \sum_{k \in C_v} w_{v,k} x_{v,k} + \sum_{k \in \mathcal{C}} c_k y_k$$

s.a.

$$x_{i,k} + x_{j,k} \leq y_k, \qquad \forall (i,j) \in E, \forall k \in C_i \cap C_j \tag{3.1}$$

$$x_{i,k} \leq y_k, \qquad \forall i \in V, \forall k \in C_i \tag{3.2}$$

$$\sum_{k \in C_v} x_{v,k} \geq 1, \qquad \forall v \in V \tag{3.3}$$

$$x_{i,k} \in \mathbb{B}, \qquad \forall i \in V, \forall k \in \mathcal{C} \tag{3.4}$$

$$y_k \in \mathbb{B}, \qquad \forall k \in \mathcal{C} \tag{3.5}$$

As Restrições 3.1 impedem que vértices adjacentes sejam coloridos com a mesma cor. As Restrições 3.2 permitem que um vértice seja colorido com uma cor k apenas se variável y_k possuir valor 1. As Restrições 3.3 obrigam cada vértice ser colorido com uma cor respeitando-se a restrição de lista de cores do vértice. Por fim, as Restrições 3.4 e 3.5 são restrições de integralidade.

3.1.2 Formulação estendida para o VCPDWC

Neste modelo, foi utilizada uma representação diferente para representar uma solução do problema. No modelo compacto queríamos saber com que cor cada vértice era colorido, enquanto no modelo estendido usaremos o fato de que todos os vértices pintados com uma mesma cor formam um conjunto independente de G. Portanto, a solução é formada por conjuntos independentes, que formam uma partição de $V(G)$, e o problema é diferente do modelo F1, agora desejamos saber quais conjuntos escolher e com quais cores colorí-los.

Utilizaremos a notação $\mathcal{L}$ para representar o conjunto de todos os conjuntos independentes de G e $\mathcal{L}_k$ para representar o conjunto de todos os conjuntos independentes de $G[V(k)]$. Para representar a solução, utilizaremos os grupos de variáveis λ_S^k e y_k. A variável λ_S^k é responsável por identificar se o conjunto independente S pintado com uma cor k está presente na solução. Desta forma, λ_S^k possui valor 1 se, e somente se, o conjunto de vértices S está na solução sendo colorido pela cor k. A variável y_k possui o mesmo significado que no

modelo compacto. A constante a_S^v representa quais vértices estão no conjunto independente S, sendo que $a_S^v = 1$ se e somente se $v \in S$. Para facilitar a leitura do modelo, utilizaremos ζ_S^k para representar o custo de um conjunto independente S colorido com a cor k e é dado por $\zeta_S^k = \sum_{v \in V} w_{v,k}\ a_v^S$.

$$(F2) \quad z = \min \sum_{k \in C} \sum_{S \in \mathcal{L}_k} \zeta_S^k\ \lambda_S^k + \sum_{k \in C} v_k\ y_k$$

s.a.

$$\sum_{k \in l_v} \sum_{S \in \mathcal{L}_k} a_S^v\ \lambda_S^k \geq 1, \qquad \forall v \in V \tag{3.6}$$

$$\sum_{S \in \mathcal{L}_k} \lambda_S^k \leq y_k, \qquad \forall k \in \mathcal{C} \tag{3.7}$$

$$\lambda_S^k \in \mathbb{B}, \qquad \forall k \in \mathcal{C}, \forall S \in \mathcal{L}_k \tag{3.8}$$

$$y_k \in \mathbb{B}, \qquad \forall k \in \mathcal{C} \tag{3.9}$$

As Restrições 3.6 obrigam todo vértice estar em algum conjunto independente presente na solução. As Restrições 3.7 restringem a presença dos conjuntos independentes na solução. Elas permitem que um conjunto independente esteja na solução com uma cor k somente se y_k possuir valor 1. Além disso, elas proíbem a presença de dois conjuntos independentes com a mesma cor na solução. As Restrições 3.8 e 3.9 são as restrições de integralidade.

3.1.3 Geração de colunas

Como o número de conjuntos independentes de um grafo é exponencial, o conjunto $\mathcal{L}$ tem tamanho exponencial em relação ao número de vértices de G, ou seja, o número de variáveis do modelo é exponencial em relação ao número de restrições.

Para lidar com o número exponencial de variáveis do modelo, utilizaremos o método de geração de colunas. Para detalhar o uso do método na resolução do problema, será apresentado o dual do modelo, o custo reduzido do grupo de variáveis que deseja-se gerar, uma interpretação das colunas do problema e, por fim, o modelo de geração de colunas.

Modelo do Problema Dual de (F2)

As variáveis duais π e μ são variáveis relacionadas às Restrições 3.6 e 3.7 do modelo (F2), respectivamente.

$$z = \max \sum_{v \in V} \pi_v$$

s.a.

$$\sum_{v \in V} a_S^v \ \pi_v + \mu_k \leq \zeta_S^k, \qquad \forall k \in \mathcal{C}, \forall S \in \mathcal{L} \quad (\lambda_S^k) \tag{3.10}$$

$$-\mu_k \leq v_k, \qquad \forall k \in \mathcal{C} \quad (y_k) \tag{3.11}$$

$$\pi_v \in \mathbb{B}, \qquad \forall v \in V \tag{3.12}$$

$$\mu_k \in \mathbb{B}, \qquad \forall k \in \mathcal{C} \tag{3.13}$$

As variáveis λ e y são relacionadas às Restrições duais 3.10 e 3.11.

Custo reduzido

Como queremos gerar variáveis do grupo λ, estamos interessados no custo reduzido correspondente. O custo reduzido desta variável é calculado a partir da Restrição 3.11 e é apresentado na Equação 3.14.

Selecionar tanto o conjunto independente quanto a cor que ele deve ser colorido acarretaria em um modelo matemático não-linear, o que não é de nosso interesse neste trabalho. Para resolver este problema, resolve-se um problema de *pricing* para cada cor k, ou seja, em cada problema de *pricing*, k torna-se uma constante.

Uma decisão de projeto foi adicionar todas as colunas com custo reduzido negativo ao PMR.

O custo reduzido de uma variável λ_S^k é apresentado na seguinte equação:

$$\overline{\zeta_S^k} = \zeta_S^k - \sum_{v \in V} a_S^v \ \pi_v - \mu_k, \forall k \in \mathcal{C}, \forall S \in \mathcal{L}. \tag{3.14}$$

$$\begin{aligned}
\text{Portanto, } pric(k) &= \min_{S \in \mathcal{L}} \overline{\zeta_S^k} \\
&= \min_{S \in \mathcal{L}} (\zeta_S^k - \sum_{v \in V} a_S^v \ \pi_v - \mu_k) \\
&= -\mu_k + \min_{S \in \mathcal{L}} (\sum_{v \in V} w_{v,k} \ a_S^v - \sum_{v \in V} a_S^v \ \pi_v) \\
&= -\mu_k + \min_{S \in \mathcal{L}} \sum_{v \in V} (w_{v,k} - \pi_v) a_S^v \\
&= -\mu_k - \max_{S \in \mathcal{L}} \sum_{v \in V} (\pi_v - w_{v,k}) a_S^v.
\end{aligned}$$

Problema de geração de colunas

No modelo clássico para o VCP o significado de uma coluna é um conjunto independente. No VCPDWC é semelhante, porém, como o custo dos conjuntos independentes que podem ser utilizados na solução para serem coloridos com uma cor dependem da cor utilizada, precisamos gerar um conjunto independente considerando-se uma determinada cor à priori.

Isso é feito executando $|C|$ problemas de *pricing*, um para cada uma das cores do problema, que consiste em encontrar um conjunto independente considerando-se o grafo induzido pelos vértices que podem ser coloridos com a cor do problema de *pricing* atual. Usaremos a notação $G[k] = (V_k, E_k)$ para denotar este grafo.

O modelo para gerar a coluna de menor custo reduzido para o VCPDWC, fixada uma cor k, é:

$$z(k) = -\mu_k - \max_{S \in \mathcal{L}} \sum_{v \in V} (\pi_v - w_{v,k}) a_S^v.$$

s.a.

$$a_i + a_j \leq 1, \qquad \forall (i,j) \in E_k \tag{3.15}$$

$$a_i \in \mathbb{B}, \qquad \forall i \in V_k \tag{3.16}$$

Como vamos fazer um *pricing* para cada cor, temos k fixado, por consequência, μ_k também torna-se fixo, daí podemos reduzir para:

$$\max_{S \in \mathcal{L}} \sum_{v \in V} (\pi_v - w_{v,k}) a_S^v$$

para garantir que o custo reduzido seja o menor possível.

Como estamos gerando o conjunto S ou seja, S é um conjunto fixo, iremos omiti-lo para evitar sobrecarga de notação no modelo.

$$(P) \quad z_k = \max \sum_{v \in V_k} ((\pi_v - w_{v,k})\ a_v)$$

s.a.

$$a_i + a_j \leq 1, \qquad \forall (i,j) \in E_k \tag{3.17}$$

$$a_i \in \mathbb{B}, \qquad \forall i \in V_k \tag{3.18}$$

Para facilitar, denotaremos o custo de um vértice na função objetivo como uma função $\overline{w} : V \times \mathcal{C} \to \mathbb{R}$ da seguinte maneira: $\overline{w}_v^k = \pi_v - w_{v,k}$.

3.2 Regras de branching

Uma das regras de *branching* mais conhecidas e usadas em algoritmos de *branch-and-price*, baseados em modelos estendidos, é a regra proposta por Ryan e Foster (1981). No entanto, essa regra de *branching* é usada em modelos de partição, nos quais as restrições são de igualdade. Uma vez que o modelo estendido proposto para o VCPDWC consiste em um modelo de cobertura, ou

seja, as restrições do modelo são inequações, a princípio, a regra de *branching* de Ryan e Foster não pode ser aplicada. Porém, se os custos de coloração dos vértices w e os custos das cores c forem positivos, é possível demonstrar que a regra de *branching* de Ryan e Foster é válida. Considere λ^* o valor das variáveis na solução ótima da relaxação linear do modelo estendido. A estratégia utilizada para selecionar um par de vértices não adjacentes para aplicar a regra de *branching* consiste em escolher um par de vértices u e v, cuja soma do valor das variáveis λ^*_S, para todo $S \in \mathcal{L}_k$ e cada cor k, cujo conjunto independente S contém simultaneamente os vértices u e v, seja o mais próximo de $0,5$.

3.3 Classes de cores

Nas instâncias do VCP, a lista de cores são idênticas para todos os vértices, ou seja, todos os vértices podem ser coloridos por qualquer cor. Desta forma, alterar as cores dos conjuntos independentes da solução não altera o valor da solução ótima. A partir dessa característica das instâncias do VCP, observamos que é possível que algumas instâncias do VCPDWC possuam cores equivalentes, no sentido em que a substituição de uma cor por outra cor equivalente não altera o valor da solução do problema nem a viabilidade da solução. Para tirar proveito dessa possibilidade, propomos uma relação de equivalência entre cores e particionamos o conjunto de cores em classe de equivalência. Desta forma, alteramos o modelo estendido para utilizar essas classes de equivalência cores ao invés do conjunto de cores para reduzir o número de problemas de *pricing* necessários para o passo de geração de colunas. Nessa seção será apresentado o conceito de classes de equivalência de cores e um modelo estendido que as utilizam.

Algumas definições são necessárias para a compreensão e formalização dessas classes.

3.3.1 Cores equivalentes

Sejam $a, b \in C$, a e b são **equivalentes** se $V(a) = V(b)$ e $w_{i,a} = w_{i,b}$ $\forall i \in V(a)$. Note que não é preciso que $c(a) = c(b)$ para que as cores sejam equivalentes nesta definição.

3.3.2 Classes de cores

Seja uma relação $R \subseteq C \times C$ definida por $R = \{(a,b) : a \in C \text{ e } b \in C \text{ e } a \text{ e } b \text{ são equivalentes}\}$. Denominaremos cada uma das classes de equivalência determinadas por R de **classe de cor**, e o conjunto de todas estas classes

por $AllClass$. Como definições auxiliares temos $rep(cl)$ sendo uma cor a que pertence à classe de cor cl, também temos $class(a) = cl \in AllClass, a \in cl$, como $AllClass$ é uma partição de $\mathcal{C}$, este valor é único. Também estenderemos a definição de V para classe de cores de forma que $V(cl) = V(rep(cl)), \forall cl \in AllClass$.

Para facilitar a leitura do modelo, denotaremos por $\mathcal{L}_{cl}$ os conjuntos independentes do grafo $G[V(cl)]$, $cl \in AllClass$ e $l_{cl}(v) = \{cl : cl \in class(a), a \in l(v)\}$. Também estenderemos nossa notação de custo reduzido de um vértice utilizando uma classe de cores como parâmetro de forma que $\overline{w}_v^{cl} = \overline{w}_v^{rep(cl)}, \forall v \in V, \forall cl \in AllClass$ e $\overline{w}_S^{cl} = \sum_{v \in S} \overline{w}_v^{cl}, \forall cl \in AllClass, \forall S \in L'_{rep(cl)}$.

3.3.3 Formulação estendida para o VCPDWC utilizando classes de cores

Utilizaremos a notação $\mathcal{L}_a$ para representar o conjunto de todos os conjuntos independentes do grafo $G[V(a)]$, $a \in \mathcal{C}$. De forma análoga, usaremos $\mathcal{L}_{cl}$ para designar os conjuntos independentes do grafo $G[V(cl)]$, $cl \in AllClass$.

Utilizando essas definições, propusemos um modelo baseado em classes de cores no qual, diferentemente do modelo F2, queremos saber com qual classe de cor um conjunto independente foi colorido. Note também que é possível uma classe de cor colorir mais do que um conjunto independente no modelo F3, mas isto é limitado a quantas cores cada classe representa.

Um modelo estendido para o VCPDWC que utiliza as classes de cores é descrito a seguir.

$$(F3) \quad z = \min \sum_{cl \in AllClass} \sum_{S \in \mathcal{L}_{cl}} \zeta_S^{rep(cl)} \, \lambda_S^{cl} + \sum_{k \in C} v_k \, y_k$$

s.a.

$$\sum_{cl \in l_{cl}(v)} \sum_{S \in \mathcal{L}_{cl}} a_S^v \, \lambda_S^{cl} \geq 1, \qquad \forall v \in V \tag{3.19}$$

$$\sum_{S \in \mathcal{L}_{cl}} \lambda_S^{cl} \leq \sum_{k \in cl} y_k, \qquad \forall cl \in AllClass \tag{3.20}$$

$$\lambda_S^k \in \mathbb{B}, \qquad \forall cl \in AllClass, \forall_S \in \mathcal{L}_{cl} \tag{3.21}$$

$$y_k \in \mathbb{B}, \qquad \forall k \in \mathcal{C} \tag{3.22}$$

3.4 Heurísticas

Nesta seção são apresentadas as heurísticas utilizadas para reduzir o tempo de execução do algoritmo de *branch-and-price*. A heurística primal é utilizada no fim da execução de cada um dos nós da árvore de *branch-and-bound* para tentar encontrar alguma solução inteira, gerando assim um limitante que pode

ser utilizado para fazer podas na árvore de *branch-and-bound*. Já a heurística de *pricing* é utilizada para encontrar colunas com custo reduzido negativo e tem tempo de execução significativamente menor que o algoritmo exato.

3.4.1 Heurística primal

Para acelerar a convergência dos limitantes, utilizamos uma heurística de arredondamento. A heurística de arredondamento é descrita no Algoritmo 1, no qual $\mathcal{L}'$ é a coleção dos conjuntos independentes do problema mestre restrito no nó atual. De maneira semelhante, $\mathcal{L}'_a$ contém os conjuntos independentes do PMR no nó atual que podem ser coloridos com a cor a, λ^* é o conjunto que possui o valor de cada uma das variáveis do problema na solução ótima do problema relaxado, no qual cada variável representa um conjunto independente de $\mathcal{L}'$. Também utilizaremos $\lambda^*(x)$ para denotar o valor na solução ótima de uma variável $x \in \mathcal{L}'$, $color^*$ é um conjunto que possui a cor de cada conjunto independente de $\mathcal{L}'$ na solução ótima e *nCores* é o numero de cores máximo do problema.

Algoritmo 1: *Rounding*

Entrada: $\mathcal{L}'$, λ^*, $color^*$, n, m, $nCores$
Saída: S

início
  $n_0 \leftarrow \{x : x \in \mathcal{L}', \lambda^*(x) = 0\}$;
  $n_1 \leftarrow \{x : x \in \mathcal{L}', \lambda^*(x) = 1\}$;
  $n_{frac} \leftarrow \{x : x \in \mathcal{L}', 0 < \lambda^*(x) < 1\}$;
  $S \leftarrow n_1$;
  $covered \leftarrow \bigcup_{s \in n_1} s$;
  faça
    $bestCol \leftarrow getBestCol(covered, n_{fraq}, n_0, \lambda^*, color_{*})$;
    $covered \leftarrow covered \cup bestCol$;
    se $bestcol \neq \emptyset$ **então**
      $S \leftarrow S \cup \{bestcol\}$;
    fim
  enquanto $|covered| < n$ *e* $bestCol \neq \emptyset$;
  retorna S;
fim

O Algoritmo 2 tem como entrada os vértices já cobertos *covered*, o conjunto de colunas com valor fracionário na relaxação linear n_{fraq}, e também as com valor zero n_0, o valor destas colunas na solução λ^*, assim como as cores correspondentes na solução $color^*$. O algoritmo seleciona o melhor conjunto independente priorizando os conjuntos que têm valor fracionário na solução ótima. Caso não exista nenhum conjunto destes que possa ser adicionado à

Algoritmo 2: *getBestCol*

Entrada: $covered, n_{fraq}, n_0, \lambda^*, color^*$
Saída: $bestcol$

```
1  início
2      bestcol ← ∅;
3      bestval ← −∞;
4      i ← 0;
5      para aux_frac ∈ n_fraq faça
6          se bestval < λ*(aux_fraq)  e  aux_fraq − covered ≠ ∅ então
7              bestval ← λ*(aux_fraq);
8              bestcol ← aux_fraq;
9          fim
10     fim
11     se bestcol = ∅ então
12         para aux_0 ∈ n_0 faça
13             auxval ← CriterioN0(aux_0, covered);
14             se bestval < auxval  e  aux_0 − covered ≠ ∅ então
15                 bestval ← auxval;
16                 bestcol ← aux_0;
17             fim
18         fim
19     fim
20     retorna bestcol
21 fim
```

solução, usa-se o Algoritmo 3 para escolher algum conjunto independente do conjunto n_0.

O Algoritmo 3 é utilizado como função de peso para as colunas que possuem valor 0 na solução do problema relaxado no nó.

Algoritmo 3: *CriterioN0*

Entrada: $column, color^*, covered, nCores$
Saída: $weight$

```
1  início
2      nColumnCover ← |column − covered| ;
3      weight ← nColumnCover ∗ (color*_column − nCores);
4      retorna weight;
5  fim
```

3.4.2 Heurística de pricing

O problema de *pricing* é resolvido de forma exata, usando-se o modelo (P), descrito na Seção 3.1.3. Como o problema de *pricing* é NP-difícil, para melhorar o desempenho do algorimo *branch-and-price*, foi utilizada uma heurística para tentar gerar colunas de custo reduzido negativo. Em cada nó da árvore

de *branch-and-price*, o método da geração de colunas é inicialmente resolvido com a heurística de *pricing*. Somente nos casos em que a heurística falha, o algoritmo exato baseado no modelo (P) é usado.

O Algoritmo 4 é uma implementação do algoritmo Greedy Randomized Adaptive Search Procedures (GRASP) proposto por Feo e Resende (1995). A escolha do algoritmo é motivada pela sua capacidade de gerar várias colunas diferentes, por causa da pseudoaleatoriedade na fase de construção dos conjuntos independentes. O Algoritmo 5 é utilizado no Algoritmo 4 assim como no Algoritmo 6 e são os passos *ConstructGreedyRandomizedSolution* e *LocalSearch* do algoritmo **GRASP**, respectivamente.

Algoritmo 4: *GRASP*

Entrada: $G, \overline{w}, crit, \alpha, nIterations$
Saída: S

```
1  início
2      S ← ∅;
3      para cl ∈ AllClass faça
4          it ← 0;
5          S_cl ← ∅;
6          enquanto it < nIterations faça
7              stable_set ← geraConjuntoIndependente(G[cl], w̄, α, cl);
8              stable_set ← localSearch(G[cl], stable_set, w̄, cl, crit);
9              S_cl ← S_cl ∪ {stable_set} ;
10             it ← it + 1
11         fim
12         S_cl ← Seleciona os 5 conjuntos de S_cl que têm maior peso w̄_s^cl, para
           s ∈ S_cl S ← S ∪ S_cl ;
13     fim
14     retorna S;
15 fim
```

O Algoritmo 4 tem como entrada um grafo G, os custos reduzidos dos conjuntos e suas respectivas classes de cores $\overline{w}$, um parâmetro α utilizado para determinar o quão guloso o Algoritmo 5 será, o critério da busca local *crit* e o número de iterações *nIterations*. O algoritmo retorna uma coleção de conjuntos independentes, com no máximo 5 conjuntos independentes para cada classe de cor, conforme descrito na linha 12 do Algoritmo 4.

O Algoritmo 5 recebe, além dos parâmetros já explicados anteriormente para o Algoritmo 4, a classe de cores cl e retorna um conjunto independente S.

Como entrada do Algoritmo 6 temos, além dos parâmetros já explicados anteriormente para o Algoritmo 4, um conjunto independente s. Este algoritmo retorna um conjunto independente. Para a busca local, implementamos duas variantes, uma que repete a busca local sempre que conseguir melhorar

Algoritmo 5: *geraConjuntoIndependente*

Entrada: $G, \overline{w}, \alpha, cl$
Saída: S

início
  $C \leftarrow V$;
  $S \leftarrow \emptyset$;
  enquanto $|C| > 0$ **faça**
    $min \leftarrow mnimo(\{\overline{w}_v^{cl}, v \in C\})$;
    $max \leftarrow mximo(\{\overline{w}_v^{cl}, v \in C\})$;
    $limiar \leftarrow (1-\alpha) \times min + \alpha \times max$;
    $RCL \leftarrow \{v : v \in C, \overline{w}_v^{cl} \geq limiar\}$;
    $v \leftarrow$ escolhe um vértice de RCL de forma aleatória;
    $S \leftarrow S \cup \{v\}$;
    $C \leftarrow C - N[v]$;
  fim
  retorna S;
fim

a solução e uma que realiza a busca local apenas uma vez. Para escolher a variante de melhoria única, basta selecionar $crit = first$ na entrada do Algoritmo 4 e, para escolher a outra variante, basta selecionar qualquer outro valor para $crit$.

Algoritmo 6: *local_search*

Entrada: $G, s, k, crit$
Saída: S

início
  $s^* \leftarrow s$;
  para $u \in s$ **faça**
    $R \leftarrow V(G) - s$;
    $S \leftarrow s - u$;
    para $v \in R$ **faça**
      $t \leftarrow S \cup \{v\}$;
      se t *é conjunto independente e* $\overline{w}_{s^*}^k < \overline{w}_t^k$ **então**
        se $crit = first$ **então**
          retorna t;
        fim
        $s^* \leftarrow t$;
      fim
    fim
  fim
  retorna s^*;
fim

CAPÍTULO

4

Experimentos Computacionais

Neste capítulo são apresentados o ambiente computacional, as instâncias testes utilizadas e alguns detalhes de implementação para discutir os experimentos empíricos realizados para avaliar os modelos e os algoritmos propostos. Por fim, uma análise dos resultados computacionais é apresentada. Resultados computacionais detalhados em cada instância são apresentados nos apêndices.

4.1 Ambiente computacional

Os testes foram executados em uma máquina Intel Core i7-4790 de 3,6GHz com 32GB de memória RAM usando o SCIP versão 3.2.1, desenvolvido por Gamrath et al. (2016), como resolvedor inteiro em conjunto com o CPLEX 12.6.1, como resolvedor linear.

4.2 Detalhes de implementação

Foi implementado um algoritmo de *branch-and-price* utilizando a linguagem de programação C e alguns recursos do SCIP. O algoritmo exato do problema *pricing* e as heurísticas foram implementados como rotinas de *callbacks* do SCIP. As regras de *branching* de Ryan e Foster (1981) também foram implementadas usando-se os recursos disponíveis no SCIP. Para a realização dos testes computacionais, foi implementado um gerador de instâncias pseudoaleatórias com parâmetros descritos na seção a seguir.

4.3 Instâncias

Para este problema, é proposto um novo conjunto de instâncias, gerado de forma pseudoaleatória utilizando-se os parâmetros descritos a seguir.

As instâncias geradas para os experimentos possuem 20, 30, 40, 50, 75 e 100 vértices. Para cada instância define-se a densidade e o número de cores baseado no número de vértices da instância conforme segue.

As densidades são configuradas como 30%, 40%, 50%, 60% ou 70% do número de arestas do grafo completo e o número de cores disponíveis como 30%, 50% ou 70% do número de vértices. Para cada combinação foram produzidas três instâncias.

O tamanho das listas de cores, o custo de colorir cada vértice com uma determinada cor e o custo de utilizar cada cor foi determinado de maneira aleatória respeitando um valor mínimo e um máximo. O tamanho da lista de cores foi definido como no mínimo 30% e no máximo 90% do número de cores disponível, o de colorir cada vértice com uma determinada cor foi definido como no mínimo 1 e no máximo 10, o custo de utilizar cada cor foi definido no mínimo como 10 e no máximo como 20.

4.4 Testes computacionais

Para avaliar e eficácia dos nossos algoritmos foram criadas 8 configurações, habilitando ou desabilitando funcionalidades. As configurações possuem três parâmetros relevantes, h, s e r. O parâmetro h é referente à utilização da heurística de *pricing*, assumindo valor 0 quando não é utilizada e valor 1 quando utilizada. O parâmetro s se refere à busca local da implementação do GRASP, assumindo valor 0 quando a busca local não é ativada, 1 quando ativamos a busca local *first improvment* e 2 quando ativamos a busca local *best improvment*. O parâmetro s diferente de 0 só faz sentido se h é diferente de 0. O parâmetro r é referente à ativação da heurística de arredondamento, assumindo valor 0 quando está desativada e valor 1 com sua ativação.

Inicialmente, realizamos testes preliminares para descartar instâncias fáceis. Para essa finalidade, consideramos instâncias com diferentes números de vértices. Na Tabela 4.1 são descritos os testes preliminares utilizando instâncias com 20, 30, 40 e 50 vértices e desabilitando-se a busca local. Os tempos reportados correspondem à média dos tempos, em segundos e o gap médio foi calculado a partir do limitante inferior L_b, dado pela solução ótima da relaxação linear, e o limitante superior U_b, obtido pelo valor de uma solução

inteira, conforme a seguinte fórmula:

$$GAP = (U_b - L_b)/L_b.$$

Configuração	Tempo médio	Gap médio
h0-s0-r0	21,6	0%
h1-s0-r1	21,2	0%
h1-s0-r0	16,9	0%
h1-s0-r1	17,0	0%

Tabela 4.1: Testes preliminares em instâncias com até 50 vértices.

Como pode ser observado na Tabela 4.1, todas as instâncias com até 50 vértices foram resolvidas de forma ótima e a um baixo custo computacional. A partir dessa análise, geramos 90 instâncias com 75 e 100 vértices, usando-se as configurações descritas na Seção 4.3.

Todas as 90 instâncias foram resolvidas com várias configurações distintas do algoritmo proposto. Na Tabela 4.2 exibimos os resultados segmentados pelas configurações comparadas. Em todos os casos, 25 instâncias dentre as de 75 vértices foram resolvidas por todas as variantes. As colunas da tabela representam o tempo médio das instâncias resolvidas na otimalidade, o gap médio das instâncias não resolvidas, o número de instâncias não resolvidas nas quais não se encontrou o limitante superior e o número de vezes que a melhor solução foi encontrada pela heurística primal (quando utilizada). A cada separação da tabela comparações foram realizadas para encontrar qual seria a melhor configuração para utilizarmos, então inicialmente foi leita uma comparação apenas utilizando o parâmetro h variando, escolhemos a configuração com h com valor 1, após isso, o parâmetro s, referente à busca local teve variação, e após a escolha do parâmetro s como melhor, o parâmetro r foi comparado, com isso obtivemos que nessas instâncias com melhor desempenho foram, a heurística habilitada, a busca local *first improvment* e a heuristica de arredondamento habilitada.

Configuração	Tempo médio	Gap médio	Sem gap	Heurística
h0-s0-r0	774,01	1,90%	8	-
h1-s0-r0	685,99	1,78%	6	-
h1-s0-r0	685,99	1,78%	6	-
h1-s1-r0	687,76	1,94%	5	-
h1-s2-r0	687,78	1,85%	9	-
h1-s1-r0	687,76	1,94%	5	-
h1-s1-r1	693,73	5,04%	0	18

Tabela 4.2: Comparação dos resultados das configurações testadas.

A primeira linha corresponde à configuração sem o uso de heurísticas e a

segunda linha, com o uso da heurística de *pricing* sem busca local. Pode-se observar que a heurística foi efetiva, reduzindo o tempo médio de solução, o número de instâncias sem gap e até o gap médio. No segundo bloco de três linhas, comparam-se as duas estratégias de busca local com a versão sem busca local. Pode-se observar que a versão de busca 1 foi melhor sucedida, encontrando o limite superior para mais uma instância. No bloco final, avalia-se o efeito do uso da heurística primal na melhor versão da heurística de *pricing*. O uso da heurística permitiu encontrar uma solução primal para todas as instâncias.

Par.		Tempo	Gap médio	Resolvidas	Heurística
D	0,3	585,36	1,03%	7	0
	0,4	650,06	1,24%	5	0
	0,5	772,58	1,24%	6	0
	0,6	1065,50	1,97%	3	1
	0,7	540,88	0,93%	4	1
c	0,3	610,67	1,77%	7	1
	0,5	805,19	1,13%	8	1
	0,7	662,71	1,04%	10	0

Tabela 4.3: Análise das execuções na melhor configuração para 75 vértices.

Par.		Tempo	Gap médio	Resolvidas	Heurística
D	0,3	-	7,43%	0	4
	0,4	-	5,60%	0	2
	0,5	-	6,31%	0	3
	0,6	-	8,93%	0	5
	0,7	-	5,09%	0	2
c	0,3	-	10,49%	0	10
	0,5	-	4,54%	0	3
	0,7	-	4,99%	0	3

Tabela 4.4: Análise das execuções na melhor configuração para 100 vértices.

Nas Tabelas 4.3 e 4.4 apresentam-se os resultados obtidos usando a melhor configuração (h1-s1-r1) nas instâncias de 75 e 100 vértices, respectivamente. As colunas da tabela indicam o tempo médio, em segundos, para as instâncias resolvidas, o gap médio para instâncias não resolvidas, a quantidade de instâncias resolvidas e o número de instâncias que a heurística primal encontrou a melhor solução. Cada tabela apresenta um resumo das instâncias agrupadas por densidade no primeiro bloco e por número de cores disponíveis no segundo bloco.

Com respeito à densidade, nota-se que as instâncias com densidade 0,6 são mais difíceis em ambos os casos, contudo para as instâncias de 100 vértices notoriamente a densidade 0,3 também teve resultados piores. Com relação ao número de cores, nota-se claramente que a redução do número de

cores dificulta a solução do problema. Também observa-se que a heurística primal apresenta maior importância no caso das instâncias maiores, no qual nenhuma delas foi resolvida.

CAPÍTULO

5

Conclusões

Foi proposto o problema da coloração de vértices com pesos dissonantes e restrições de cores, uma generalização do conhecido problema de coloração de vértices com uma aplicação prática e um algoritmo exato de *branch-and-price*, baseado em geração de colunas. Para a melhoria do tempo de execução do algoritmo de geração de colunas, foi utilizada uma heurística GRASP, que foi selecionada dada a pseudoaleatoriedade do algoritmo, gerando assim mais diversidade nos conjuntos independentes gerados. Também foi proposta uma heurística primal de arredondamento para encontrar soluções inteiras e limitantes primais. Para decidir o grupo de instâncias relevante para a avaliação do algoritmo foram feitos testes preliminares em instâncias de tamanho 20, 30, 40 e 50. Testes preliminares indicaram que essas instâncias eram fáceis e passamos a considerar instâncias maiores. Para a avaliação do algoritmo, foram propostas 90 instâncias geradas de maneira pseudoaleatória. Por meio dos experimentos, foi possível classificar as instâncias pelo nível de dificuldade em relação à densidade do grafo e ao total de cores. Dadas as configurações foi possível identificar que o problema torna-se significativamente mais difícil quando aumentamos o número de vértices de 75 para 100. Durante os testes verificou-se que a maior parte do tempo de execução do algoritmo foi durante o *pricing*. Desta forma, uma potencial melhoria consiste em avaliar outros algoritmos exatos e heurísticas para o problema de *pricing*.

Referências Bibliográficas

Bazaraa, M. e Jarvis, J. (1977). *Linear Programming and Network Flows*. Board of Advisor Engineering. Wiley. Citado na página 9.

Bondy, J. A. e Murty (2008). *Graph theory*. Springer. Citado na página 2.

Feo, T. A. e Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of global optimization*, 6:109–133. Citado na página 26.

Furini, F. e Malaguti, E. (2012). Exact weighted vertex coloring via branch-and-price. *Discrete Optimization*, 9(2):130–136. Citado na página 7.

Gamrath, G., Fischer, T., Gally, T., Gleixner, A. M., Hendel, G., Koch, T., Maher, S. J., Miltenberger, M., Müller, B., Pfetsch, M. E., Puchert, C., Rehfeldt, D., Schenker, S., Schwarz, R., Serrano, F., Shinano, Y., Vigerske, S., Weninger, D., Winkler, M., Witt, J. T., e Witzig, J. (2016). The SCIP Optimization Suite 3.2. Technical report, Optimization Online. Citado na página 29.

Gross, J. L. e Yellen, J. (2004). *Handbook of graph theory*. CRC press. Citado na página 2.

Lucci, M., Nasini, G., e Severín, D. (2019). A branch and price algorithm for list coloring problem. *Electronic Notes in Theoretical Computer Science*, 346:613–624. The proceedings of Lagos 2019, the tenth Latin and American Algorithms, Graphs and Optimization Symposium (LAGOS 2019). Citado na página 17.

Ryan, D. M. e Foster, B. A. (1981). An integer programming approach to scheduling. In Wren, A., editor, *Computer scheduling of public transport urban passenger vehicle and crew scheduling*, páginas 269–280. North-Holland, Amsterdam. Citado nas páginas 21 e 29.

Ternier, I. (2017). *Exact algorithms for the Vertex Coloring Problem and its generalisations. (Résolution exacte du Problème de Coloration de Graphe et*

ses variantes). PhD thesis, PSL Research University, Paris, France. Citado na página 7.

Wolsey, L. (1998). *Integer Programming*. Wiley Series in Discrete Mathematics and Optimization. Wiley. Citado na página 9.

Wu, Q. e Hao, J.-K. (2012). An effective heuristic algorithm for sum coloring of graphs. *Computers & Operations Research*, 39(7):1593–1600. Citado na página 7.

Zykov, A. A. (1949). On some properties of linear complexes. *Matematicheskii sbornik*, 66(2):163–188. Citado na página 6.

APÊNDICE

A

Resultados computacionais detalhados

Neste apêndice são descritos os resultados computacionais detalhados por instância. A Tabela A.1 apresenta os tempos de execução, em segundos, para cada uma das configurações descritas na Seção 4.4. Representa-se por TLE a situação em que não foi encontrada a solução ótima em determinada instância no tempo limite do teste. O nome das instâncias refletem suas características separadas por hífen, sendo o primeiro valor o número de vértices, o segundo a densidade, o terceiro o número de cores e o último o índice dessa instância, já que são criadas mais do que uma instância com as mesmas características.

A organização da Tabela A.2 é a mesma que a Tabela A.1 e apresenta o gap final para cada uma das instâncias em cada uma das configurações descritas na Seção 4.4.

Tabela A.1: Tabela com o tempo de execução, em segundos, de cada instância nas configurações testadas

Instâncias	**h0-s0-r0**	**h0-s0-r1**	**h1-s0-r0**	**h1-s0-r1**	**h1-s1-r0**	**h1-s2-r0**	**h1-s1-r1**
75-0.3-22-0	337.2	226.1	206.8	196.3	149.3	154.7	139.3
75-0.3-22-1	90.5	135.4	104.3	91.8	76.2	123.1	69.8
75-0.3-22-2	1,010.2	948.9	996.8	946.3	999.5	845.0	983.2
75-0.3-38-0	TLE	TLE	TLE	TLE	TLE	TLE	TLE
75-0.3-38-1	866.6	863.3	712.7	1,115.1	789.2	948.2	667.0
75-0.3-38-2	1,122.8	991.2	1,046.3	1,037.1	970.6	1,119.0	1,114.7
75-0.3-52-0	TLE	TLE	TLE	TLE	TLE	TLE	TLE
75-0.3-52-1	727.2	640.9	645.6	686.1	779.5	626.8	808.7
75-0.3-52-2	282.3	252.9	258.5	321.2	356.3	404.5	314.8
75-0.4-22-0	645.0	659.9	657.8	652.9	602.8	646.4	644.0
75-0.4-22-1	TLE	TLE	TLE	TLE	TLE	TLE	TLE
75-0.4-22-2	673.8	662.0	565.5	532.9	690.6	768.8	612.7
75-0.4-38-0	TLE	TLE	TLE	TLE	TLE	TLE	TLE
75-0.4-38-1	691.8	680.5	874.7	798.2	830.8	894.7	822.3
75-0.4-38-2	552.3	384.2	373.7	326.2	390.5	351.8	373.9
75-0.4-52-0	TLE	TLE	TLE	TLE	TLE	TLE	TLE
75-0.4-52-1	1,273.1	1,282.8	923.6	1,068.8	847.4	828.3	797.5
75-0.4-52-2	TLE	TLE	TLE	TLE	TLE	TLE	TLE
75-0.5-22-0	542.7	574.1	482.6	513.1	514.9	467.0	505.2
75-0.5-22-1	1,536.2	1,517.0	1,358.5	1,394.9	1,255.5	1,300.0	1,320.5
75-0.5-22-2	TLE	TLE	TLE	TLE	TLE	TLE	TLE

continua na próxima página

Tabela A.1 – continuação da página anterior

Instâncias	h0-s0-r0	h0-s0-r1	h1-s0-r0	h1-s0-r1	h1-s1-r0	h1-s2-r0	h1-s1-r1
75-0.5-38-0	TLE	TLE	TLE	TLE	TLE	TLE	TLE
75-0.5-38-1	624.3	590.2	550.2	493.4	456.3	528.2	537.3
75-0.5-38-2	TLE	TLE	TLE	TLE	TLE	TLE	TLE
75-0.5-52-0	1,580.9	887.3	1,435.5	1,334.5	876.5	989.9	1,210.8
75-0.5-52-1	435.1	524.3	328.8	360.6	507.6	446.9	467.8
75-0.5-52-2	809.6	806.9	740.0	623.1	881.1	647.0	593.9
75-0.6-22-0	TLE	TLE	TLE	TLE	TLE	TLE	TLE
75-0.6-22-1	TLE	TLE	TLE	TLE	TLE	TLE	TLE
75-0.6-22-2	TLE	TLE	TLE	TLE	TLE	TLE	TLE
75-0.6-38-0	1,533.1	1,604.0	1,447.0	1,470.4	1,700.0	1,459.3	1,664.8
75-0.6-38-1	TLE	TLE	TLE	TLE	TLE	TLE	TLE
75-0.6-38-2	TLE	TLE	TLE	TLE	TLE	TLE	TLE
75-0.6-52-0	444.1	427.0	288.9	242.0	239.1	273.2	259.4
75-0.6-52-1	1,261.1	1,279.9	1,133.2	1,157.7	1,246.1	1,211.1	1,272.3
75-0.6-52-2	TLE	TLE	TLE	TLE	TLE	TLE	TLE
75-0.7-22-0	TLE	TLE	TLE	TLE	TLE	TLE	TLE
75-0.7-22-1	TLE	TLE	TLE	TLE	TLE	TLE	TLE
75-0.7-22-2	TLE	TLE	TLE	TLE	TLE	TLE	TLE
75-0.7-38-0	356.5	384.2	362.0	357.6	351.2	333.5	357.6
75-0.7-38-1	TLE	TLE	TLE	TLE	TLE	TLE	TLE
75-0.7-38-2	991.7	1,031.1	966.6	892.5	876.4	1,030.5	904.0
75-0.7-52-0	150.5	122.3	116.2	108.0	73.5	103.3	124.5

continua na próxima página

Tabela A.1 – continuação da página anterior

Instâncias	h0-s0-r0	h0-s0-r1	h1-s0-r0	h1-s0-r1	h1-s1-r0	h1-s2-r0	h1-s1-r1
75-0.7-52-1	811.8	827.2	574.0	569.9	732.9	693.2	777.4
75-0.7-52-2	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.3-30-0	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.3-30-1	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.3-30-2	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.3-50-0	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.3-50-1	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.3-50-2	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.3-70-0	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.3-70-1	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.3-70-2	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.4-30-0	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.4-30-1	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.4-30-2	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.4-50-0	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.4-50-1	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.4-50-2	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.4-70-0	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.4-70-1	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.4-70-2	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.5-30-0	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.5-30-1	TLE	TLE	TLE	TLE	TLE	TLE	TLE

continua na próxima página

Tabela A.1 – continuação da página anterior

Instâncias	h0-s0-r0	h0-s0-r1	h1-s0-r0	h1-s0-r1	h1-s1-r0	h1-s2-r0	h1-s1-r1
100-0.5-30-2	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.5-50-0	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.5-50-1	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.5-50-2	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.5-70-0	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.5-70-1	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.5-70-2	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.6-30-0	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.6-30-1	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.6-30-2	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.6-50-0	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.6-50-1	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.6-50-2	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.6-70-0	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.6-70-1	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.6-70-2	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.7-30-0	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.7-30-1	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.7-30-2	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.7-50-0	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.7-50-1	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.7-50-2	TLE	TLE	TLE	TLE	TLE	TLE	TLE

continua na próxima página

Tabela A.1 – continuação da página anterior

Instâncias	h0-s0-r0	h0-s0-r1	h1-s0-r0	h1-s0-r1	h1-s1-r0	h1-s2-r0	h1-s1-r1
100-0.7-70-0	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.7-70-1	TLE	TLE	TLE	TLE	TLE	TLE	TLE
100-0.7-70-2	TLE	TLE	TLE	TLE	TLE	TLE	TLE

Tabela A.2: Tabela com o gap de cada instância nas configurações testadas

Instâncias	h0-s0-r0	h0-s0-r1	h1-s0-r0	h1-s0-r1	h1-s1-r0	h1-s2-r0	h1-s1-r1
75-0.3-22-0	0.00%	0.0%	0.0%	0.00%	0.00%	0.00%	0.00%
75-0.3-22-1	0.00%	0.0%	0.0%	0.00%	0.00%	0.00%	0.00%
75-0.3-22-2	0.00%	0.0%	0.0%	0.00%	0.00%	0.00%	0.00%
75-0.3-38-0	1.18%	1.3%	1.2%	1.19%	1.24%	1.27%	1.28%
75-0.3-38-1	0.00%	0.0%	0.0%	0.00%	0.00%	0.00%	0.00%
75-0.3-38-2	0.00%	0.0%	0.0%	0.00%	0.00%	0.00%	0.00%
75-0.3-52-0	0.82%	0.8%	0.6%	0.59%	0.72%	0.79%	0.78%
75-0.3-52-1	0.00%	0.0%	0.0%	0.00%	0.00%	0.00%	0.00%
75-0.3-52-2	0.00%	0.0%	0.0%	0.00%	0.00%	0.00%	0.00%
75-0.4-22-0	0.00%	0.0%	0.0%	0.00%	0.00%	0.00%	0.00%
75-0.4-22-1	1.17%	1.2%	1.2%	1.22%	1.27%	1.18%	1.31%
75-0.4-22-2	0.00%	0.0%	0.0%	0.00%	0.00%	0.00%	0.00%
75-0.4-38-0	0.89%	0.9%	0.8%	0.90%	0.86%	0.81%	0.81%
continua na próxima página							

Tabela A.2 – continuação da página anterior

Instâncias	h0-s0-r0	h0-s0-r1	h1-s0-r0	h1-s0-r1	h1-s1-r0	h1-s2-r0	h1-s1-r1
75-0.4-38-1	0.00%	0.0%	0.0%	0.00%	0.00%	0.00%	0.00%
75-0.4-38-2	0.00%	0.0%	0.0%	0.00%	0.00%	0.00%	0.00%
75-0.4-52-0	1.26%	1.3%	1.3%	1.31%	1.66%	1.33%	1.31%
75-0.4-52-1	0.00%	0.0%	0.0%	0.00%	0.00%	0.00%	0.00%
75-0.4-52-2	1.47%	1.5%	1.5%	1.62%	1.55%	1.53%	1.54%
75-0.5-22-0	0.00%	0.0%	0.0%	0.00%	0.00%	0.00%	0.00%
75-0.5-22-1	0.00%	0.0%	0.0%	0.00%	0.00%	0.00%	0.00%
75-0.5-22-2	1.16%	1.9%	1.1%	1.12%	1.15%	1.90%	1.10%
75-0.5-38-0	0.84%	0.8%	0.8%	0.71%	1.21%	1.20%	1.20%
75-0.5-38-1	0.00%	0.0%	0.0%	0.00%	0.00%	0.00%	0.00%
75-0.5-38-2	1.16%	1.2%	1.7%	1.39%	1.75%	1.40%	1.42%
75-0.5-52-0	0.00%	0.0%	0.0%	0.00%	0.00%	0.00%	0.00%
75-0.5-52-1	0.00%	0.0%	0.0%	0.00%	0.00%	0.00%	0.00%
75-0.5-52-2	0.00%	0.0%	0.0%	0.00%	0.00%	0.00%	0.00%
75-0.6-22-0	2.34%	1.9%	1.8%	1.80%	1.85%	2.47%	6.31%
75-0.6-22-1	1.72%	1.7%	1.9%	1.79%	1.25%	1.96%	1.24%
75-0.6-22-2	0.96%	0.9%	1.0%	0.92%	1.18%	0.91%	1.13%
75-0.6-38-0	0.00%	0.0%	0.0%	0.00%	0.00%	0.00%	0.00%
75-0.6-38-1	0.99%	1.0%	0.9%	0.96%	0.93%	0.80%	0.81%
75-0.6-38-2	1.21%	1.2%	1.4%	1.39%	1.46%	1.37%	1.44%
75-0.6-52-0	0.00%	0.0%	0.0%	0.00%	0.00%	0.00%	0.00%
75-0.6-52-1	0.00%	0.0%	0.0%	0.00%	0.00%	0.00%	0.00%

continua na próxima página

Tabela A.2 – continuação da página anterior

Instâncias	h0-s0-r0	h0-s0-r1	h1-s0-r0	h1-s0-r1	h1-s1-r0	h1-s2-r0	h1-s1-r1
75-0.6-52-2	0.89%	0.9%	0.9%	0.90%	0.96%	0.94%	0.90%
75-0.7-22-0	0.77%	0.7%	0.9%	0.84%	0.68%	0.68%	0.68%
75-0.7-22-1	0.93%	1.5%	1.0%	1.01%	1.47%	0.95%	1.05%
75-0.7-22-2	2.03%	1.8%	1.8%	7.90%	1.43%	3.88%	1.35%
75-0.7-38-0	0.00%	0.0%	0.0%	0.00%	0.00%	0.00%	0.00%
75-0.7-38-1	0.89%	0.9%	0.9%	0.88%	0.89%	0.89%	0.91%
75-0.7-38-2	0.00%	0.0%	0.0%	0.00%	0.00%	0.00%	0.00%
75-0.7-52-0	0.00%	0.0%	0.0%	0.00%	0.00%	0.00%	0.00%
75-0.7-52-1	0.00%	0.0%	0.0%	0.00%	0.00%	0.00%	0.00%
75-0.7-52-2	0.50%	0.5%	0.7%	0.49%	0.69%	0.56%	0.65%
100-0.3-30-0	3.81%	3.1%	2.5%	12.14%	2.81%	3.50%	1.28%
100-0.3-30-1	1.93%	12.4%	3.9%	11.47%	3.17%	2.59%	12.94%
100-0.3-30-2	1.82%	1.5%	1.2%	1.55%	2.18%	1.62%	13.83%
100-0.3-50-0	1.53%	1.8%	1.8%	1.63%	1.65%	1.82%	1.56%
100-0.3-50-1	1.19%	1.2%	1.3%	1.22%	1.25%	1.27%	1.23%
100-0.3-50-2	0.86%	1.7%	2.2%	1.68%	0.88%	0.83%	1.23%
100-0.3-70-0	4.90%	2.5%	3.2%	2.81%	4.16%	3.04%	22.68%
100-0.3-70-1	4.18%	9.3%	2.6%	2.53%	3.42%	2.93%	10.52%
100-0.3-70-2	1.52%	1.6%	1.8%	1.48%	1.57%	1.51%	1.57%
100-0.4-30-0	2.50%	13.2%	1.8%	8.29%	1.61%	3.43%	2.79%
100-0.4-30-1	2.27%	1.1%	1.1%	1.10%	2.99%	1.08%	1.12%
100-0.4-30-2	0.81%	0.8%	0.6%	0.67%	0.75%	0.68%	0.71%

continua na próxima página

Tabela A.2 – continuação da página anterior

Instâncias	h0-s0-r0	h0-s0-r1	h1-s0-r0	h1-s0-r1	h1-s1-r0	h1-s2-r0	h1-s1-r1
100-0.4-50-0	-	19.8%	-	19.74%	4.09%	-	22.24%
100-0.4-50-1	3.36%	13.7%	1.8%	1.76%	3.94%	3.01%	2.28%
100-0.4-50-2	0.82%	0.9%	0.8%	0.83%	0.90%	0.85%	1.03%
100-0.4-70-0	1.27%	1.3%	1.7%	1.37%	2.15%	1.28%	1.32%
100-0.4-70-1	1.66%	2.0%	1.7%	1.88%	2.31%	2.41%	2.26%
100-0.4-70-2	3.62%	2.0%	2.5%	5.66%	1.95%	3.84%	16.60%
100-0.5-30-0	-	19.4%	-	17.03%	3.49%	-	14.34%
100-0.5-30-1	1.71%	1.7%	2.7%	2.24%	3.31%	1.97%	8.98%
100-0.5-30-2	-	19.0%	-	17.48%	-	-	20.97%
100-0.5-50-0	2.01%	1.9%	1.9%	1.67%	2.03%	1.83%	1.89%
100-0.5-50-1	2.47%	2.6%	2.2%	2.19%	2.50%	1.84%	2.49%
100-0.5-50-2	2.53%	2.6%	2.5%	2.07%	2.12%	2.53%	2.08%
100-0.5-70-0	1.79%	1.8%	2.3%	2.26%	2.04%	1.81%	2.27%
100-0.5-70-1	2.62%	1.9%	2.2%	1.95%	3.06%	2.81%	2.65%
100-0.5-70-2	1.14%	1.1%	1.1%	1.14%	1.11%	1.11%	1.09%
100-0.6-30-0	-	10.0%	-	14.65%	-	-	14.76%
100-0.6-30-1	2.73%	10.7%	3.5%	7.80%	3.45%	3.69%	12.99%
100-0.6-30-2	-	16.7%	3.1%	16.60%	-	-	20.36%
100-0.6-50-0	1.81%	2.3%	2.2%	2.38%	2.44%	2.33%	2.45%
100-0.6-50-1	-	6.7%	3.1%	6.65%	3.02%	-	8.56%
100-0.6-50-2	2.78%	13.8%	2.8%	13.17%	4.13%	-	14.81%
100-0.6-70-0	1.25%	1.5%	1.2%	1.58%	1.15%	2.01%	1.82%

continua na próxima página

Tabela A.2 – continuação da página anterior

Instâncias	h0-s0-r0	h0-s0-r1	h1-s0-r0	h1-s0-r1	h1-s1-r0	h1-s2-r0	h1-s1-r1
100-0.6-70-1	1.87%	2.1%	1.7%	2.11%	2.12%	2.14%	2.16%
100-0.6-70-2	2.13%	2.5%	2.1%	2.13%	2.40%	2.45%	2.50%
100-0.7-30-0	-	15.8%	-	13.88%	-	-	18.74%
100-0.7-30-1	-	12.5%	-	8.31%	-	-	13.02%
100-0.7-30-2	0.61%	0.7%	0.5%	0.48%	0.47%	0.43%	0.48%
100-0.7-50-0	2.06%	2.0%	2.0%	1.96%	1.99%	2.01%	2.33%
100-0.7-50-1	2.54%	2.5%	1.4%	2.26%	1.21%	3.12%	2.60%
100-0.7-50-2	1.90%	1.7%	1.3%	1.29%	1.35%	1.31%	1.30%
100-0.7-70-0	2.66%	2.7%	2.4%	2.19%	2.18%	2.78%	2.14%
100-0.7-70-1	5.55%	7.7%	2.8%	2.63%	2.47%	2.57%	2.53%
100-0.7-70-2	2.24%	3.5%	1.8%	2.02%	2.45%	2.50%	2.68%